2020

Dynamic Scene Recognition

Xiaoming Peng
University of Wollongong

UNIVERSITY
OF WOLLONGONG
AUSTRALIA

Dynamic Scene Recognition

Xiaoming Peng

This thesis is presented as part of the requirements for the conferral of the degree:

Doctor of Philosophy

Supervisor:
Prof. Abdesselam Bouzerdoum

Co-supervisors:
Assoc. Prof. Son Lam Phung

The University of Wollongong
School of Electrical, Computer and Telecommunications Engineering

June, 2020

This research has been conducted with the support of an Australian Government Research Training Program Scholarship.

Declaration

I, *Xiaoming Peng*, declare that this thesis is submitted in partial fulfilment of the requirements for the conferral of the degree *Doctor of Philosophy*, from the University of Wollongong, is wholly my own work unless otherwise referenced or acknowledged. This document has not been submitted for qualifications at any other academic institution.

Xiaoming Peng

June 10, 2020

Abstract

Many applications need to deal with a scene that evolves over time. This thesis addresses the *dynamic scene recognition* problem. Video-based dynamic scene recognition methods and mid-level representations are comprehensively reviewed, followed by several proposed methods. First, a part-based approach for dynamic scene recognition is presented. In this approach, representative parts are located by clustering local convolutional features extracted using a pre-trained Fast R-CNN model. A set cover problem is then formulated to select the discriminative parts, which are subsequently refined. Local features from a video segment can be extracted and aggregated. Extensive experiments show that this approach is very competitive with state-of-the-art methods that use global features, achieving an overall accuracy of 91.5% and 98.3%, respectively, on two benchmark datasets—the Maryland and the Yupenn.

Then, a trajectory-based approach for dynamic scene recognition is proposed. Densely and evenly distributed trajectories are extracted from a video segment. Deep-learned features are extracted from the trajectories to form feature sequences, which are fed into a Long-Short-Term-Memory (LSTM) network (called the local LSTM) to learn their temporal behavior. The local LSTM is trained with synthetic trajectory data instead of real trajectory data, to avoid the impurity issue and the imbalance issue with the real training data. Similarly, a global LSTM is also trained to extract features from whole video frames. The proposed

approach achieved an overall accuracy of 88.5% and 98.3%, respectively, on the Maryland and the Yupenn. Moreover, to locate category-dependent discriminative trajectories in a video segment, an optimization problem is formulated to jointly learn the discriminative part detectors of all categories.

Three methods are proposed to infer the input to the generator of auxiliary classifier generative adversarial networks (ACGANs). The first two methods, named i-ACGAN-r and i-ACGAN-d, are "inverting" methods, which obtain an inverse mapping from an image to the class label and the latent sample. By contrast, the third method, referred to as i-ACGAN-e, directly infers both the class label and the latent sample by introducing an encoder into an ACGAN. These methods were evaluated on three test datasets (Intel, Place8 Static and Place8 Dynamic), using two performance measures: the class recovery accuracy and the image reconstruction error. i-ACGAN-e achieved about 5% more class recovery accuracy than the other two methods on Intel and Place8 Static, and 2.6% more class recovery accuracy on Place8 Dynamic. However, the images generated by the other two methods have smaller image reconstruction errors.

Finally, a wide-baseline stereo matching approach is proposed. In this approach, a per-column cost matrix (PCC) is combined with a feature-vector-based weighting strategy to achieve both matching accuracy and computational efficiency. The proposed approach is applied with the DAISY feature descriptor. For the Fountain and HerzJesu Dataset, the proposed method yields a middle rank in terms of precision and recall rates when compared with two other methods. However, it is 2.42 times and 1.98 times faster than the other two methods, respectively. For the 2014 Middlebury Stereo Dataset, the proposed method achieved an average disparity estimation accuracy of 0.624, higher than those obtained by the other two methods.

Acknowledgments

This is my *second* PhD—I received my first one in 2005, 15 years ago. At that time I was a young man in my early 30s, full of passion and energy, just started my career at a Chinese university. I had been working for more than ten years there and finally got promoted to the position of the Associate Professor. I lived a relatively comfortable life and was quite used to it—until one day I decided to make a change. I decided to challenge myself by pursing another PhD degree overseas. Studying abroad had always been one of my dreams, but I failed to recognize its importance and necessity at a younger age. Now I was over 40; this idea seemed so crazy and unrealistic. I had many sleepless nights, asking myself a barrage of questions, like,"Are you able to pass the IELTS test? Are you able to secure a scholarship? Are you really prepared to give up everything you have obtained so far? What if you cannot finish your study in time? After graduation, will you find a decent job?..." Now reflecting back on these past four years, the most challenging time for this PhD study is actually the day I made up my mind. Since then, I had met many difficulties and challenges, but I knew there was no turning back. Fortunately, I never gave up. I am privileged to have this invaluable experience that will shape the rest of my life.

I would like to thank, first and foremost, my principal supervisor Senior Prof. Abdesselam Bouzerdoum for his endless support. He offered me a scholarship to make this dream come true. Without his financial support I was not able

to make it to Australia in the first place. He had always been very considerate through out this whole period of my study, trying to help me in every respect as best as he could. His help began even before I came to Wollongong. Though a very busy man, he managed to spare the time to help me prepare the application documents. When I just arrived, he brought me warmth at the cold nights by sending me a blanket and a nightstand. Meanwhile, he is a very tolerant and patient gentleman, spending many hours discussing with me and reviewing and correcting my papers. There was a period of hard time that I could not make a progress in my research, but his confidence in me and encouragement had been unwavering. I wish I could have done better to pay back his efforts and support.

I am very grateful to my co-supervisor Associate Prof. Dr. Son Lam Phung. He is a passionate and very hardworking gentleman. He works so hard that I noticed some emails he sent me were at one or two o'clock in the morning. While Prof. Abdesselam Bouzerdoum was on leave, he took on the responsibility of leading the whole group, on top of his already very busy teaching and research timetable. Meanwhile, he is also a rigorous person and always maintains a high standard of research and publication. I benefited a lot from working with him in revising my papers. He patiently pointed out my mistakes and taught me a load of useful writing skills. Not only was he very helpful in my research, but he also taught me how to face challenges and form an optimistic attitude towards life. I have a lot to learn from him, including his passion, hardworking attitude and enthusiasm.

I am very honored and privileged to have the opportunity to work with the fellow members in our research group. They are beautiful people more than happy to help each other. I will always remember their names: Fok Tivive, Anh Nguyen, Xiaolin Tang, Soan Duong, Thanh Le Hoang, Paul Ang, and Saeed Ghorbani. I also met friends who offered me great help to make life easier, they are: Yuxi Ruan, Yu Liu, Chengpu Duan, and Xianwei Huang. I would also like to thank the staff of the School of Electrical, Computer and Telecommunications

Engineering (SECTE) and Faculty of Engineering and Information Sciences (EIS) for giving me personal and professional support. My gratitude to the University of Wollongong is unreserved. Not only did it offer me a second chance, but also waived my tuition fees. I am very proud to be a member of its alumni.

Many thanks to Prof. Mohammed Bennamoun from the University of Western Australia, who had been very helpful whenever I turned to him for advice. He encouraged me to pursue my dream and helped me seek prospective supervisors. His reference letters played an important role in my enrollment into the University of Wollongong.

I owe my family for their unconditional support. My wife was originally disgruntled with my decision to go overseas for a "pointless work" (in her own words), but finally came to terms with it. She traveled several times to Australia in the past several years to boost my morale, but most of the time she had to work and live alone in China. I will use the rest of my life to compensate for her sacrifice. My son lived with me for two years in Wollongong. During that period of time, a solid father-son bond was forged as the "one and a half men" learned to support each other and cope with difficulties together. My mother came to see me by flying alone for eleven hours in economy class at the age of 71. During her stay, she took on the responsibility of cooking so that I ate well and more importantly, was able to contribute more time to work. She did the best she could to help me. Finally, my big brother and younger sister had been constantly encouraging me, ensuring me that they would take good care of other family members while I was away. Upon the finalization of this thesis, I am very relieved to see they finally made it through the lock-down of the City of Wuhan, the hardest-hit city in China since the COVID-19 outbreak. My heart is always with them.

Contents

List of Figures

List of Tables

Acronyms

2D — two dimensional

3D — three dimensional

AAEs — adversarial autoencoders

ACGANs — auxiliary classifier generative adversarial networks

ALI — adversarial learned inference

ARMA — autoregressive moving average

BN — batch normalization

BoF — bag-of-features

BoS — bag-of-systems

BOSS — bag-of-SFA-Symbols

BoW — bag-of-words

CCCP — Concave-Convex Procedure

CNN — convolutional neural network

DALI — decomposed adversarial learned inference

DPM — deformable part models

DT — dense trajectories

EM — expectation-maximum

EMD — Earth Mover's Distance

GANs — generative adversarial networks

GC — graph cuts

GMM — Gaussian Mixture Model

GRU — gated recurrent unit

HoF — histograms of optical flow

HoG — histograms of oriented gradients

iDT — improved dense trajectories

IFV — Improved Fisher Vector

KDE — kernel density estimation

LBP — loopy belief propagation

LDA — Latent Dirichlet Allocation

LDA — linear discriminant analysis

LDS — linear dynamical system

LLC — Locality-constrained Linear Coding

LP — linear program

LSTM — Long-Short-Term-Memory

LTFF — long-term frequency feature

MBH — motion boundary histogram

MLP — multi-layer perceptrons

MPLBP — max-product loopy belief propagation

MRFs — Markov Random Fields

MSE — mean-square error

NCC — normalized cross correlation

NN — nearest neighbor

OB — object bank

PAM — Partitioning Around Medoids

PCA — principal component analysis

PCC — per-column cost

PDF — probability distribution function

pLSA — probabilistic Latent Semantic Analysis

R-CNN — regional convolutional neural network

ReLU — Rectified Linear Units

RNN — recurrent neural network

ROIs — regions of interest

SAD — sum of absolute differences

SCP — set cover problem

SFA — slow feature analysis

SPM — spatial pyramid matching

SSD — sum of squared differences

SVM — support vector machine

TAD — truncated absolute difference

TSN — temporal segment networks

VAEs — variational autoencoders

VLAD — Vector of Locally Aggregated Descriptors

Chapter 1

Introduction

Chapter contents

1.1 Research motivation

Many applications—from automatic monitoring of forest fire [1] to autonomous driving [2] and abnormal event detection [3]—need to deal with a scene that evolves over time. This thesis addresses the *dynamic scene recognition* problem. Different applications require different types of data to recognize a dynamic scene. For forest fire monitoring [1] as well as abnormal event detection applications [3, 4], monocular videos are sufficient. Most efforts of this research are devoted to video-based scene recognition. Video-based dynamic scene classification was first introduced in the context of human action recognition, which proved that scene context information can help improve action recognition [5].

Specifically, given a monocular video, the task is to recognize or classify the semantic label of the scene captured in the video, *e.g.*, avalanches, rushing rivers, highways. However, in some applications such as autonomous driving, dynamic scene recognition may also be performed in a 3D scenario. One preliminary sub-task of 3D dynamic scene recognition is depth estimation. Once the depth information is available, a 3D model of the scene can be reconstructed and used for subsequent sub-tasks such as scene categorization, object detection and tracking, and event categorization. Thus, in view of the promising prospect of 3D dynamic scene recognition, some efforts of this research are dedicated to stereo matching for depth estimation, which is a preliminary step towards a future complete 3D dynamic scene recognition system.

If static scene recognition is already challenging due to the large range of intra-class variations within each image category, including scale, view point and illumination changes, video-based dynamic scene recognition is more involving for the following three reasons. First, data redundancy is much higher in the frames from a video than in a set of unrelated images. Thus, more training data are required for video recognition than for image recognition. Second, the inherent dynamics of a scene can be entangled with a moving camera, making it difficult to describe the temporal behavior of the scene. Third, the extra temporal dimension causes much higher computational cost.

1.1.1 Research objectives

The objective of this research is to explore the following research aspects:

1. Explore better feature descriptors of video features, including more compact spatial/spatio-temporal representations of video clips.

2. Investigate the role of mid-level feature representations in dynamic scene recognition. This is inspired by the success of mid-level feature representations, part-based models in particular, in image recognition applications.

3. Explore the potential of generative models for scene recognition. Most existing methods for scene recognition are based on discriminative models. However, generative models address this problem in another direction.

4. Explore wide-baseline stereo matching, which serves as a prerequisite step towards 3D dynamic scene recognition.

1.2 Research contributions

The main contributions of this research are described as follows:

- The development of a part-based dynamic scene recognition method that comprises three main novelties. First, after representative parts are automatically learned from a training set, a set cover problem (SCP) is formulated to select the discriminative parts. Second, refining the selected parts is realized by fine-tuning a Fast R-CNN model instead of designing a complex optimization scheme. Third, a new technique is proposed to temporally aggregate local fully-connected layer features within a video segment.

- The development of a trajectory-based dynamic scene recognition method that comprises two main novelties. First, the local LSTM network in this method is trained with synthetic trajectory feature sequences instead of real ones. This is because it is non-trivial to determine the label for a real trajectory feature sequence. The synthetic feature sequences are generated with the aid of a series of GANs. Second, to locate category-dependent discriminative trajectories in a video segment, an optimization problem is formulated to simultaneously learn the discriminative part detectors of all categories. A secondary novelty with the proposed method is an approach to generate densely and evenly distributed trajectories across a video segment.

- The design and comparison of three methods to infer the input to the generator of ACGANs. ACGANs are a type of conditional GANs. Compared

with regular GANs, ACGANs use class labels along with the latent samples as input to the generator. This property enables ACGANs to be used as a promising reconstruction-based classifier whereby an image can be classified according to the inferred class label.

- The development of a cost aggregation method for wide-baseline stereo matching, where a PCC matrix is combined with a feature-vector-based weighting strategy to achieve both matching accuracy and computational efficiency.

1.3 Publications

Following is the list of publications based on this PhD research project, which took place from late July 2016 to February 2020.

1. X. Peng, A. Bouzerdoum, S. L. Phung, "Infer the input to the generator of auxiliary classifier generative adversarial networks," 2020 IEEE International Conference on Image Processing (ICIP), accepted.
2. X. Peng, A. Bouzerdoum, "Part-based feature aggregation method for dynamic scene recognition," The International Conference on Digital Image Computing: Techniques and Applications (DICTA), Perth, Australia, 2019, pp. 1–8.
3. X. Peng, A. Bouzerdoum, S. L. Phung, "Efficient cost aggregation for feature-vector-based wide-baseline stereo matching," EURASIP Journal on Image and Video Processing, doi:10.1186/s13640-018-0249-y, 2018.
4. X. Peng, A. Bouzerdoum, S. L. Phung, "An efficient local method for stereo matching using daisy features," 2017 IEEE International Conference on Image Processing (ICIP), pp. 2503–2507.

Papers under review:

5. X. Peng, A. Bouzerdoum, S. L. Phung, "A part-based spatial and temporal

aggregation method for dynamic scene recognition," invited submission to Neural Computing and Applications (NCAA) Special Issue on DICTA 2019. This paper is now under revision.

6. X. Peng, A. Bouzerdoum, S. L. Phung, "A trajectory-based method for dynamic scene recognition," International Journal of Pattern Recognition and Artificial Intelligence.

1.4 Thesis structure

This thesis is structured as follows:

- **Chapter 1** describes the research project, its objectives, a summary of related publications, and the structure of the thesis.
- **Chapter 2** provides a detailed literature review for video-based dynamic scene recognition. Some general video classification approaches are also included in this chapter to provide a wider picture of relevant tools and techniques.
- **Chapter 3** provides a comprehensive literature review for mid-level feature representations. This chapter along with **Chapter 2** offers the background knowledge for **Chapter 4** and **Chapter 5**.
- **Chapter 4** describes a part-based dynamic scene recognition approach. This method spatially and temporally aggregates local features extracted from local regions of a video segment.
- **Chapter 5** describes a trajectory-based dynamic scene recognition approach. This method aggregates information extracted from trajectories that are distributed densely and evenly across a video segment. In addition, category-specific discriminative trajectories are located in a video segment.

- **Chapter 6** describes three methods to infer the input to the generator of ACGANs. The three methods are compared with two performance measurs, the class recovery accuracy and the image reconstruction error, on three test datasets.

- **Chapter 7** describes a cost aggregation in which a PCC matrix is combined with a feature-vector-based weighting strategy to achieve both matching accuracy and computational efficiency. The method is applied with the DAISY feature descriptor for wide-baseline stereo matching.

- **Chapter 8** summarizes the research findings and provides concluding remarks and future directions.

Chapter 2

Review of Dynamic Scene Recognition Methods

Chapter contents

In this chapter, methods for video-based dynamic scene recognition will be reviewed. The methods to be discussed and analyzed are grouped into two broad categories, handcrafted methods and deep-learning methods, according to whether they use manually engineered features or deep-learned features. Each

category is further divided into several subcategories. This chapter, along with **Chapter 3**, provides the literature review for **Chapter 4** and **Chapter 5**.

2.1 Handcrafted methods

Handcrafted methods use manually engineered features to represent a video. These handcrafted features are further divided into three types: 1) spatio-temporal features, 2) motion features, and 3) manifold representations. Spatio-temporal features are mostly obtained by convolving a video with 3D filters. Motion features are built upon motion models, trying to explicitly extract motion information from videos. Manifold-representation-based methods regard features extracted from video frames as points in a Riemannian manifold, and investigate how to measure the similarity between these points.

2.1.1 Spatio-temporal features

A series of dynamic scene recognition methods is based on the so-called "spatio-temporal" features [6, 7, 8, 9, 10, 11]. Given a video $V(x,y,t)$, where (x,y,t) is the coordinates of a 3D point inside the video, the spatio-temporal feature $E_{\mathrm{ST}}(x,y,t)$ at (x,y,t) is extracted by convolving V with a 3D Gaussian third derivative filter $G^{(3)}_{\mathrm{3D}}$, defined as:

$$G^{(3)}_{\mathrm{3D}}(x,y,t;\boldsymbol{\theta},\sigma) = \kappa \frac{\partial^3}{\partial \boldsymbol{\theta}^3} \exp(-\frac{x^2+y^2+t^2}{\sigma^2}), \tag{2.1}$$

where κ is a normalization factor, $\boldsymbol{\theta}$ denotes the orientation of the 3D filter, and σ the scale at which the convolution is applied. The convolution is carried out as

$$E_{\mathrm{ST}}(x,y,t;\boldsymbol{\theta},\sigma) = \sum_{\Omega} \left| (G^{(3)}_{\mathrm{3D}} * V)(x,y,t) \right|^2, \tag{2.2}$$

where Ω is a small neighborhood around (x,y,t) and the symbol * denotes the convolution operation.

In the frequency domain, motion occurs at a plane through the origin. For example, if all the frames of the video are subjected to a consistent uniform translation $\mathbf{n} = \{u,v,1\}$ along the image plane, *i.e.* $V(x,y,t) = V(x-u,y-v,t-1)$, then the spectrum $\tilde{V}(\omega_x,\omega_y,\omega_t)$ of V can be given as

$$\tilde{V}(\omega_x,\omega_y,\omega_t) = \mathcal{F}(V)\delta(\omega_x u + \omega_y v + \omega_t), \tag{2.3}$$

where $(\omega_x,\omega_y,\omega_t)$ is the frequency coordinates, $\mathcal{F}(V)$ denotes the Fourier transform of V, and $\delta(\cdot)$ is the Dirac delta function which is one only when $\omega_x u + \omega_y v + \omega_t = 0$ and zero otherwise. Obviously, $\mathbf{n} = \{u,v,1\}$ is the normal of a plane passing through the origin in the frequency domain, and the spectrum $\tilde{V}(\omega_x,\omega_y,\omega_t)$ is restricted to this plane, denoted as $\mathbf{\Pi}(\mathbf{n})$. For any other $\mathbf{n}$, only $(N+1)$ equally spaced directions along $\mathbf{\Pi}(\mathbf{n})$ need to be considered for a Gaussian derivative filter of order N [12]. These $(N+1)$ directions $\boldsymbol{\theta}_{\mathbf{\Pi}(\mathbf{n})} = \left\{\boldsymbol{\theta}^i_{\mathbf{\Pi}(\mathbf{n})}\right\}_{0\le i\le N}$ are given as

$$\boldsymbol{\theta}^i_{\mathbf{\Pi}(\mathbf{n})} = \cos\left(\frac{2\pi i}{N+1}\right)\boldsymbol{\theta}_\mathrm{a}(\mathbf{n}) + \sin\left(\frac{2\pi i}{N+1}\right)\boldsymbol{\theta}_\mathrm{b}(\mathbf{n}), 0 \le i \le N, \tag{2.4}$$

where $\boldsymbol{\theta}_\mathrm{a}(\mathbf{n}) = \frac{\mathbf{n}\times\mathbf{e}_x}{\|\mathbf{n}\times\mathbf{e}_x\|}$, $\boldsymbol{\theta}_\mathrm{b}(\mathbf{n}) = \mathbf{n}\times\boldsymbol{\theta}_\mathrm{a}(\mathbf{n})$, and $\mathbf{e}_x$ denotes the unit vector along the ω_x axis. The measurements of $E_\mathrm{ST}(x,y,t;\boldsymbol{\theta},\sigma)$ for all $\boldsymbol{\theta} \in \boldsymbol{\theta}_{\mathbf{\Pi}(\mathbf{n})}$ are then aggregated to generate temporal energies $E_\mathrm{T}(x,y,t;\mathbf{n},\sigma)$:

$$E_\mathrm{T}(x,y,t;\mathbf{n},\sigma) = \sum_{\boldsymbol{\theta}\in\boldsymbol{\theta}_{\mathbf{\Pi}(\mathbf{n})}} E_\mathrm{ST}(x,y,t;\boldsymbol{\theta},\sigma). \tag{2.5}$$

In the methods of [6, 7, 8, 9, 10, 11], seven $\mathbf{n}$'s are considered: 1) static $\mathbf{n}_\mathrm{s} = (0,0,1)^\mathrm{T}$, 2) rightward $\mathbf{n}_\mathrm{r} = (1,0,1)^\mathrm{T}$, 3) downward $\mathbf{n}_\mathrm{d} = (0,-1,1)^\mathrm{T}$, 4) leftward $\mathbf{n}_\mathrm{l} = (-1,0,1)^\mathrm{T}$, 5) upward $\mathbf{n}_\mathrm{u} = (0,1,1)^\mathrm{T}$, 6) horizontal flicker $\mathbf{n}_\mathrm{hf} = (1,0,0)^\mathrm{T}$, and 7) vertical flicker $\mathbf{n}_\mathrm{vf} = (0,1,0)^\mathrm{T}$. As a result, seven measurements of $E_\mathrm{T}(x,y,t;\mathbf{n},\sigma)$ are computed

at position (x, y, t) and scale σ for these $\mathbf{n}$'s. These measurements are further $\ell 1$-normalized as follows to provide additional invariance to multiplicative photometric variations:

$$\hat{E}_{\mathrm{T}}(x, y, t; \mathbf{n}, \sigma) = \frac{E_{\mathrm{T}}(x, y, t; \mathbf{n}, \sigma)}{\sum_{\mathbf{n}} E_{\mathrm{T}}(x, y, t; \mathbf{n}, \sigma) + \epsilon}, \mathbf{n} \in \{\mathbf{n}_{\mathrm{s}}, \mathbf{n}_{\mathrm{r}}, \mathbf{n}_{\mathrm{d}}, \mathbf{n}_{\mathrm{l}}, \mathbf{n}_{\mathrm{u}}, \mathbf{n}_{\mathrm{hf}}, \mathbf{n}_{\mathrm{vf}}\}, \tag{2.6}$$

where ϵ is a small constant introduced to explicitly capture the lack of oriented spatio-temporal structures.

The spatial feature has proven to be complimentary to the spatio-temporal feature [9, 11]. It is extracted at individual frames of a video sequence, using two dimensional (2D) Gaussian first derivative filters defined as

$$G^{(1)}_{\mathrm{2D}}(x, y; \theta, \sigma) = \kappa \frac{\partial}{\partial \theta} \exp(-\frac{x^2 + y^2}{\sigma^2}), \tag{2.7}$$

where κ is again a normalization factor. The spatial orientation θ takes eight values: $\theta = i\frac{\pi}{4}$, $(0 \leq i \leq 7)$. The spatial energies are computed by convolving a frame $I(x, y)$ with $G^{(1)}_{\mathrm{2D}}(x, y; \theta, \sigma)$ at a given spatial orientation θ and scale σ:

$$E_{\mathrm{S}}(x, y; \theta, \sigma) = \sum_{\Omega} \left| (G^{(1)}_{\mathrm{2D}} * I)(x, y) \right|^2, \tag{2.8}$$

where Ω is a small neighborhood around (x, y).

The primitive features obtained above can be used to construct histograms and paired with a nearest-neighbor (NN) classifier for classification. For example, $7 \times 4 = 28$ measurements of $E_{\mathrm{ST}}(x, y, t; \boldsymbol{\theta}, \sigma)$ are computed for a small neighborhood Ω of a video for the seven normals $\{\mathbf{n}_{\mathrm{s}}, \mathbf{n}_{\mathrm{r}}, \mathbf{n}_{\mathrm{d}}, \mathbf{n}_{\mathrm{l}}, \mathbf{n}_{\mathrm{u}}, \mathbf{n}_{\mathrm{hf}}, \mathbf{n}_{\mathrm{vf}}\}$; the whole video is then represented by a histogram of these 28 measurement values [8]. Some commonly-used distance measures between two histograms include the $\ell 1$ and $\ell 2$ distances, Bhattacharyya coefficients, χ^2 statistics, and the Earth Mover's Distance (EMD) [13]. More sophisticated encoding methods, such as the Locality-constrained Linear Coding (LLC) [14] and the Improved Fisher Vector (IFV) [15],

can also be used to encode these primitive features into higher-level features. The final features can be used as input to a conventional classifier, such as the support vector machine (SVM) and random forests [16]. Besides the spatio-temporal feature and the spatial feature discussed above, some studies also incorporated chromatic information (*e.g*. CIE-LUV colors) as a complementary feature [9, 11].

Quan *et al*. developed a sparse coding scheme to represent the spatio-temporal structure of raw video data [17]. An alternating approach was designed to iteratively solve for the sparse tensor and update the dictionaries. The dictionaries can be learned in a fast and scalable way because of their separability and orthogonality. Once they are learned, feature maps (which are sparse representations) of a video can be extracted and selected to form a histogram-based descriptor, which is then fed into a classifier.

2.1.2 Motion modeling

The methods presented in this category try to explicitly extract motion information from videos. Marszalek *et al*. [5] proposed a bag-of-features (BoF) video representation. In their method, first, salient 3D and 2D corners are extracted using the 3D Harris detector [18] and the 2D Harris detector [19]. Then, in each region surrounding a 3D corner, histograms of oriented gradients (HoG) features [20] and histograms of optical flow (HoF) features [21] are computed to capture the dynamic appearance and motion patterns in the region. To capture the static appearance, the SIFT features [22] are computed in each region surrounding a 2D corner. For each of these three features (HoG, HoF and SIFT), a visual vocabulary is created such that a video sample is represented as an order-less distribution of visual words, called bag-of-features [23]. This method relies heavily on successful detection of the salient 3D and 2D corners. Vasudevan *et al*. designed a five-dimensional motion flow vector (5DMFV) to locate dynamic regions in a video [24]. The 5DMFV is composed of optical flow field, its divergence, and

the gradient of the magnitude of the vector field. This method is susceptible to the quality of the computed optical flow field. Shroff *et al*. represented a video as a combination of three chaotic invariants: the Lyapunov exponent, the correlation integral, and the correlation dimension [25]. The Lyapunov exponent characterizes the level of chaos in the system, whereas the correlation integral and the correlation dimension estimate the system complexity. In their method, a 960-dimensional GIST descriptor [26] is first extracted at each video frame. Each dimension of the GIST descriptor forms an individual time series. A 10-dimensional vector comprising the three chaotic invariants is then obtained for each time-series. The video is finally represented by the concatenation of these 10-dimensional vectors.

Some methods investigate the role of linear dynamical system (LDS) in modeling the motions of a video [27, 28, 29]. A video of N frames, denoted $\{I_t \in \mathbb{R}^d\}_{t=1}^N$, where each frame is represented as a d-dimensional vector, can be modeled as the output of an LDS as follows [27]:

$$\begin{cases} \mathbf{z}_{t+1} &= \mathbf{A}\mathbf{z}_t + \mathbf{B}\mathbf{v}_t \\ I_t &= \mathbf{m} + \mathbf{C}\mathbf{z}_t + \mathbf{w}_t, \end{cases} \tag{2.9}$$

where $\mathbf{z}_t \in \mathbb{R}^n$ is the n-dimensional *hidden state* at time t, matrix $\mathbf{A} \in \mathbb{R}^{n \times n}$ models the dynamics of $\mathbf{z}_t$, matrix $\mathbf{C} \in \mathbb{R}^{d \times n}$ maps the hidden state to the output of the LDS, vector $\mathbf{m} = \frac{1}{N}\sum_{t=1}^{N} I_t$, and $\mathbf{w}_t$ and $\mathbf{B}\mathbf{v}_t$ are the measurement and process noise, respectively. The order of the LDS is n. Under the LDS model, the video $\{I_t \in \mathbb{R}^d\}_{t=1}^N$ is characterized by $\{\mathbf{A}, \mathbf{C}\}$. The similarity between two LDSs can be measured using subspace angles [30]. Ravichandran *et al*. proposed a bag-of-systems (BoS) [28], in which LDS features replace conventional features in the BoF framework. One issue with the LDS features, however, is that they do not lie in the Euclidean space. To address this issue, a lower dimensional representation of the LDS features is first obtained, which lies in the Euclidean space, then K-

means clustering can be applied to measure the similarity between two dimension-reduced LDS features. Huang *et al*. addressed the problem of sparse coding and dictionary learning with LDS [29]. One problem with LDS-based methods is that the LDS is built on the first-order Markov property and linearity assumption; thus its ability to describe complex scene dynamics is intrinsically limited.

Thériault *et al*. [31] advocated the slow feature analysis (SFA) [32, 33] as an alternative to optical flow and LDS representations. The SFA transforms a temporal input signal into a new representation that varies slowly but retains relevant information of the original input signal. Mathematically, given a d-dimensional input signal $\mathbf{x}(t) = [x_1(t), x_2(t), ..., x_d(t)]$, a set of K real-valued functions $\{g_1(\mathbf{x}), g_2(\mathbf{x}), ..., g_K(\mathbf{x})\}$ is sought such that each output signal $y_j(t) = g_j(\mathbf{x}(t))$ $(1 \leq j \leq K)$ formulates the following optimization problem [33]:

$$\min \langle \dot{y}_j^2 \rangle_t, \tag{2.10}$$

under the following constraints:

$$\langle y_j \rangle_t = 0 \text{ (zero mean)} \tag{2.11}$$

$$\langle y_j^2 \rangle_t = 1 \text{ (unit variance)} \tag{2.12}$$

$$\forall i < j, \langle y_i y_j \rangle_t = 0 \text{ (decorrelation and order)}, \tag{2.13}$$

where $\langle \cdot \rangle$ and $\dot{y}$ denote the time averaging operation and the time derivative of y, respectively. This problem can be transformed into the generalized eigenvalue problem:

$$\langle \dot{\mathbf{x}} \dot{\mathbf{x}}^{\mathrm{T}} \rangle_t \mathbf{W} = \langle \mathbf{x} \mathbf{x}^{\mathrm{T}} \rangle_t \mathbf{W} \boldsymbol{\Lambda}, \tag{2.14}$$

where $\mathbf{W}$ is the matrix of the generalized eigenvectors and $\boldsymbol{\Lambda}$ is the diagonal matrix of the generalized eigenvalues $\{\lambda_1, \lambda_2, ..., \lambda_d\}$. The K eigenvectors $\{\mathbf{w}_1, \mathbf{w}_2, ..., \mathbf{w}_K\}$ $(K \leq d)$ that correspond to the K smallest generalized eigenvalues define the output signals $y_j(t) = \mathbf{w}_j^{\mathrm{T}} \mathbf{x}$. Features of any type, including cuboids of pixels clipped

from video frames [34], can be used as the input signal. The SFA achieved a moderate recognition accuracy of 74% on the Maryland dataset [25] with the V1 feature [35]. Some recently proposed multivariate time series analysis methods, such as Bag-of-SFA-Symbols (BOSS) [36], WEASEL (Word ExtrAction for time SEries cLassification) [37], and WEASEL+MUSE (MUltivariate Symbols and dErivatives) [38] may also be used for motion modeling.

Besides the above methods that are dedicated to dynamic scene recognition, some methods on human activity recognition may also be applicable in this field. An excellent survey on these methods was conducted in [39]. Among them, the dense trajectories (DT) method [40] and the improved dense trajectories (iDT) method [41] were state-of-the-art before the emergence of deep learning methods. Based on the optical flow constraints, the DT method obtains dense trajectories by tracking a grid of pixels across the video frames. Along these dense trajectories, four features are extracted: the trajectory shape, HoG, HoF, and the motion boundary histogram (MBH) [42]. These four features are combined into a global feature to represent the video segment. The iDT method improves upon the DT method by removing camera motions from the scene motions. Recently, Khokher *et al*. proposed a tensor decomposition method to combine the HoG feature and the MBH feature extracted from the refined dense trajectories [43]. Papadopoulos *et al*. proposed the so-called "localized trajectories" as an improved version of the dense trajectories, where motion trajectories are clustered around human body joints provided by RGB-D cameras and then encoded by local BoW [44]. However, this method still uses traditional handcrafted features (the trajectory shape, HoG, HoF and MBH) to describe a trajectory. By contrast, the proposed trajectory-based dynamic scene recognition approach present in **Chapter 5** uses deep-learned features to describe a trajectory.

2.1.3 Manifold representation

The majority of the methods discussed above are limited to the Euclidean space. However, some studies look at the non-Euclidean space [45, 46, 47]. Harandi *et al*. formulated the video classification problem as an image set matching problem, in which classification is realized by comparing the probability distribution functions (PDFs) of image sets [45]. These PDFs can be modeled using Riemannian manifolds. Let $\mathcal{X} = \{\mathbf{x}_i\}_{i=1}^{N}$ be a set of N images, where each $\mathbf{x}_i$ is a feature extacted from an image. A PDF on $\mathcal{X}$ is a function $p : \mathcal{X} \to \mathbb{R}^+$ such that $\int_{\mathcal{X}} p(\mathbf{x})d\mathbf{x} = 1$. Let $\mathcal{M}$ be a family of PDFs on $\mathcal{X}$. With certain assumptions (*e.g*. differentiability), $\mathcal{M}$ forms a Riemannian manifold. In [45], the kernel density estimation (KDE) was used to obtain an estimation $\hat{p}(\mathbf{x})$ of the PDF:

$$\hat{p}(\mathbf{x}) = \frac{1}{N\sqrt{|2\pi\boldsymbol{\Sigma}|}} \sum_{i=1}^{N} \exp\Big[-\frac{1}{2}(\mathbf{x}-\mathbf{x}_i)^{\mathrm{T}}\boldsymbol{\Sigma}^{-1}(\mathbf{x}-\mathbf{x}_i)\Big], \tag{2.15}$$

where $\boldsymbol{\Sigma}$ is the covariance matrix of $\mathcal{X}$, and $|2\pi\boldsymbol{\Sigma}|$ denotes the determinant of $2\pi\boldsymbol{\Sigma}$. Harandi *et al*. advocated using the Csiszár f-divergence over the Fisher-Rao metric to measure the similarity between two PDFs p and q [45]. The Csiszár f-divergence δ_f is defined as

$$\delta_f(p\|q) = \int f\Big(\frac{p}{q}\Big)dq, \tag{2.16}$$

where f is a convex function on $(0,\infty)$ with $f(1) = 0$. Different choices of f yield different metrics. In particular, they used the *Hellinger distance* and the *Jeffrey divergence* (also called the *symmetric KL divergence*), which are two special cases of the Csiszár f-divergence. Both metrics are invariant under differentiable and invertible transformations (diffeomorphisms). Invariance to diffemorphism is an attractive property in image set matching, as images in a set can be subjected to many variations. A query image set is classified according to its distance to the nearest-neighbor training image set. Harandi *et al*. extended the framework of

sparse coding and dictionary learning from the Euclidean space to the Grassmann manifold [46]. Faraki *et al*. [47] proposed the Riemannian version of the conventional Vector of Locally Aggregated Descriptors (VLAD) [48], called R-VLAD. For a set of features $\{\mathbf{x}_i\}_{i=1}^{N}$ extracted from a video, and a codebook $\{\mathbf{c}_j\}_{j=1}^{M}$, where M is the size of the codebook, the conventional VLAD representation is a $d \times M$ matrix $\mathbf{V}$, whose jth column is computed as

$$\mathbf{V}(:, j) = \sum_{\mathbf{x}_i : \mathrm{NN}(\mathbf{x}_i) = \mathbf{c}_j} (\mathbf{x}_i - \mathbf{c}_j), \tag{2.17}$$

where $\mathrm{NN}(\mathbf{x}_i)$ denotes the code word that is nearest to the feature $\mathbf{x}_i$. In the R-VLAD representation, the geodesic distance is used to measure the similarity between an inquiry feature manifold and a code word manifold. The counterpart of the vector subtraction operation in Eq. (2.17) is the local difference vector in the R-VLAD representation. One issue with manifold-representation-based methods, however, is the temporal information in the video is neglected.

Finally, the performances of some handcrafted methods are listed in Table 2.1 to enable a comparison between them. The statistics in the table are copied from the corresponding references that reported the results obtained by these methods on two benchmark dynamic scene datasets, the Maryland dataset [25] and the Yupenn dataset [7]. It can be seen from the table that manifold representations are slightly better than the other two types of methods. On the other hand, spatio-temporal features generally outperform motion modeling methods.

Table 2.1: Recognition accuracy results (in %) obtained by some handcrafted methods on two benchmark dynamic scene datasets.

	Spatio-temporal			**Motion modeling**			**Manifolds**	
Dataset	SOE [7]	CSO [9]	BoSE [10]	5DMFV [24]	Chaotic [25]	SFA [31]	R-M [45]	R-V [47]
Maryland	42.0	67.7	77.7	-	52.3	74.6	80.2	-
Yupenn	79.0	86.0	96.2	85.6	-	85.0	-	99.8

2.2 Deep-learning methods

The current state-of-the-art techniques for dynamic scene recognition leverage the power of deep learning [49, 50, 51, 52]. This section presents and discusses deep-learning-based methods. Note that in addition to the approaches that are dedicated to dynamic scene recognition, this section also includes some methods that were developed for the more broad aim of video classification. The methods presented in this section are grouped into four types: 1) statistical moments, 2) deep-learned spatio-temporal features, 3) recurrent neural networks (RNNs), and 4) two-stream CNNs.

2.2.1 Statistical moments

Several methods use statistical moments to aggregate the information in a video segment [49, 50, 53]. Qi *et al*. developed a so-called Transferred ConvNet Feature (TCoF) scheme to recognize both dynamic textures and dynamic scenes [49]. It has two levels. The first level extracts spatial features (called s-TCoF) from each video frame using the AlexNet model [54, 55], which was fine-tuned using the dynamic scene data. The second level of TCoF computes the first and second-order statistics over the s-TCoF features to generate temporal features. For a set of N s-TCoF features, $\{\mathbf{x}_i\}_{i=1}^N$, its first-order statistic is the mean vector

$$\mathbf{u} = \frac{1}{N}\sum_{i=1}^{N}\mathbf{x}_i, \tag{2.18}$$

whereas the second-order statistic of $\{\mathbf{x}_i\}_{i=1}^N$ is the covariance matrix

$$\mathbf{S} = \frac{1}{N}\sum_{i=1}^{N}(\mathbf{x}_i - \mathbf{u})(\mathbf{x}_i - \mathbf{u})^{\mathrm{T}}. \tag{2.19}$$

The authors of [49] argued that the correlation between two different dimensions of $\mathbf{x}_i$ is relatively low and therefore less informative; they only used the diagonal

elements of $\mathbf{S}$, $\mathbf{v} = \text{diag}(\mathbf{S})$, as an approximation. The video-level representation is the concatenation of $\mathbf{u}$ and $\mathbf{v}$.

Hong *et al.*'s D3 method [50] deployed a pre-trained VGG model [56] to extract the convolutional layer features, which are used as the spatial feature representation. The VGG model was pre-trained on the ImageNet dataset [57]. Central moments were adopted to aggregate the spatial features within a video segment to generate the temporal feature representation. The D3 method has two feature representations, S_{conv} and T_{conv}, to describe the static appearance and temporal behavior of a video, respectively. The S_{conv} feature for a video segment of N frames is expressed as

$$S_{\text{conv}} = [\mathbf{s}_1^1, \mathbf{s}_1^2, ..., \mathbf{s}_1^{49}, ..., \mathbf{s}_i^1, \mathbf{s}_i^2, ..., \mathbf{s}_i^{49}, ..., \mathbf{s}_N^1, \mathbf{s}_N^2, ..., \mathbf{s}_N^{49}]. \tag{2.20}$$

In Eq. (2.20), the features $\{\mathbf{s}_i^1, \mathbf{s}_i^2, ..., \mathbf{s}_i^{49}\}$ are the conv5_3 features extracted from the ith ($1 \le i \le N$) frame, where each $\mathbf{s}_i^n$ ($1 \le n \le 49$) is a local feature of 512 dimensions. The conv5_3 feature corresponds to the "conv5_3" layer of the VGG model. The T_{conv} feature is computed by accumulating the local features in S_{conv} using the central moments:

$$T_{\text{conv}} = [\mathbf{t}^1, \mathbf{t}^2, ..., \mathbf{t}^{49}], \tag{2.21}$$

where $\mathbf{t}^n = \frac{1}{N}\sum_{i=1}^{N} \|\mathbf{s}_i^n - \mathbf{u}^n\|_2^2$, and $\mathbf{u}^n = \frac{1}{N}\sum_{i=1}^{N} \mathbf{s}_i^n$. Both the S_{conv} and T_{conv} features are encoded using the IFV [15] and concatenated to form the D3 feature. The S_{conv} and T_{conv} features were constructed using only *key frames* and *key segments* in [50], where the key frames were selected from all the frames of a video using a clustering method; then, a key segment was formed using the frames around each key frame. Similarly, Gangopadhyay *et al*. used a pre-trained CNN model to extract a fully connected layer feature as the spatial feature [53]. They used four moments—the mean, standard deviation, skewness and kurtosis—to aggregate the spatial features to obtain the temporal feature.

One issue with statistical moments is that they are intrinsically order-less,

completely neglecting the temporal order of the frames in a video segment. Thus they may not be sufficient to represent the inherent dynamics of a video.

2.2.2 Deep-learned spatio-temporal features

Some researches intend to learn the spatio-temporal representation of video segments directly from the raw data using a 3D CNN [52, 58, 59, 60, 61, 62]. The 3D convolutions are performed in the 3D CNNs by convolving 3D kernels with a video segment. Mathematically, for a 3D CNN, the value $v_{ij}^{x,y,z}$ at position (x,y,z) on its jth feature map of the ith convolutional layer is given by [58]

$$v_{ij}^{x,y,z} = \tanh\left(b_{ij} + \sum_{m}\sum_{p=1}^{P_i}\sum_{q=1}^{Q_i}\sum_{r=1}^{R_i} w_{ijm}^{pqr} v_{(i-1)m}^{(x+p)(y+q)(z+r)}\right), \tag{2.22}$$

where $\tanh(\cdot)$ is the hyperbolic tangent function, defined as $\tanh(x) = \frac{e^{2x}-1}{e^{2x}+1}$, b_{ij} is the bias for this feature map, m indexes over the set of feature maps in the previous $(i-1)$th layer, $v_{(i-1)m}^{(x+p)(y+q)(z+r)}$ is the value at the $(x+p, y+q, z+r)$ position on the mth feature map of the $(i-1)$th layer, w_{ijm}^{pqr} is the corresponding value at the 3D kernel, and P_i, Q_i and R_i are the dimensions of the 3D kernel. Karpathy *et al*. [59] investigated three approaches to fusing the temporal information of a video segment: 1) early fusion, 2) late fusion, and 3) slow fusion, see Figure 2.1 for illustrations.

The early fusion approach directly feeds a stack of video frames into the first convolutional layer. The aim is to obtain the temporal information immediately at the pixel level. The late fusion approach, however, postpones the fusion of two frames (which are the beginning and ending frames of a video segment), until at the first fully-connected layer. The rationale for this approach is the global motion of the video segment can be captured by comparing the output of two streams, each of which processes one frame. The slow fusion approach adopts a hierarchical architecture such that the lowest layers process sub-segments of the

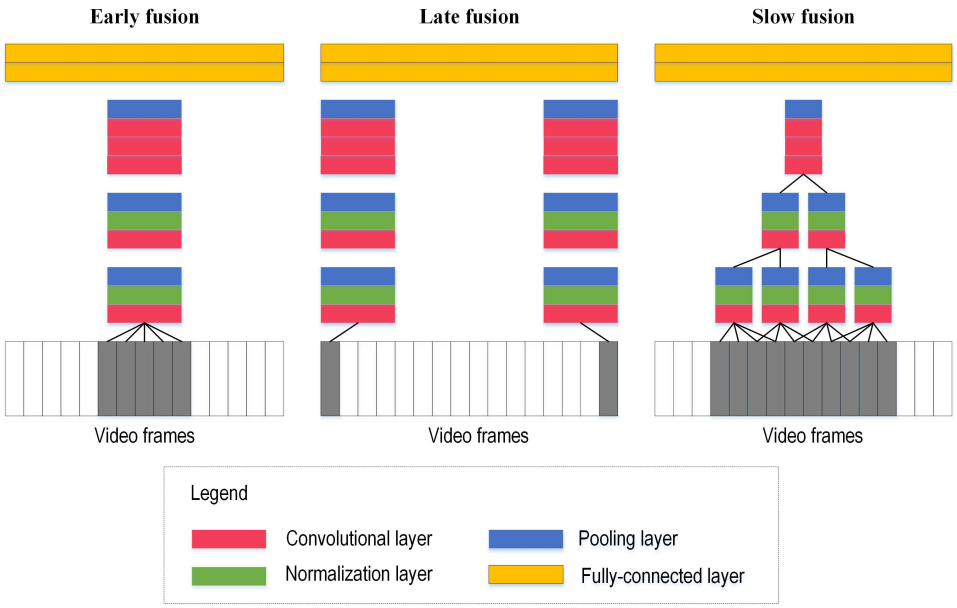


Figure 2.1: Three approaches to fusing the temporal information of a video segment [59].

video frames, and the information of these sub-segments is gradually fused at higher layers. This approach is a balance between the previous two approaches, and outperforms both of them. Arunnehru *et al*. applied a 3D CNN on the so-called "3D motion cuboids" for action detection and recognizing in videos [63]. A 3D motion cuboid is formed by stacking up frame differences. Applying a 3D CNN on the motion cuboids instead of directly on the original video frames is based on the presumption that motion information is better encoded on frame differences. However, this presumption may not hold when camera motions are entangled with the scene motions. By contrast, the proposed part-based dynamic scene recognition approach present in **Chapter 4** aggregates information from discriminative parts of a video segment, rather than relying on motions across the frames.

Feichtenhofer *et al*. extended the ResNet [64] to develop the temporal residual network [52]. The ResNet allows for constructing a very deep network by using the "shortcut connections", which skip one or more layers of the network to connect the output of a preceding layer to a succeeding layer. The building blocks of the ResNet are comprised of a few stacked layers. The role of these building

blocks is to fit a residual mapping instead of a desired underlying mapping. The hypothesis behind this strategy is that it is easier to optimize the residual mapping than to optimize the original underlying mapping directly. Formally, denoting the underlying mapping as $\mathcal{H}(\mathbf{x})$, where $\mathbf{x}$ is the input to a building block, the building block fits the residual mapping of $\mathcal{F}(\mathbf{x}) := \mathcal{H}(\mathbf{x}) - \mathbf{x}$. Then the underlying mapping can be recast into $\mathcal{F}(\mathbf{x}) + \mathbf{x}$. The "bottleneck" building block for a ResNet is shown in Figure 2.2 (a). The bottleneck building block consists of three convolutional layers, where the two 1×1 convolutional layers reduce and then restore the dimensions of feature maps, while the 3×3 convolutional layer forms a bottleneck with smaller feature map dimensions. The temporal residual network in [52] incorporates a 3-tap temporal filter and a channel-wise affine scaling weight into the building block of ResNet, as shown in Figure 2.2 (b).

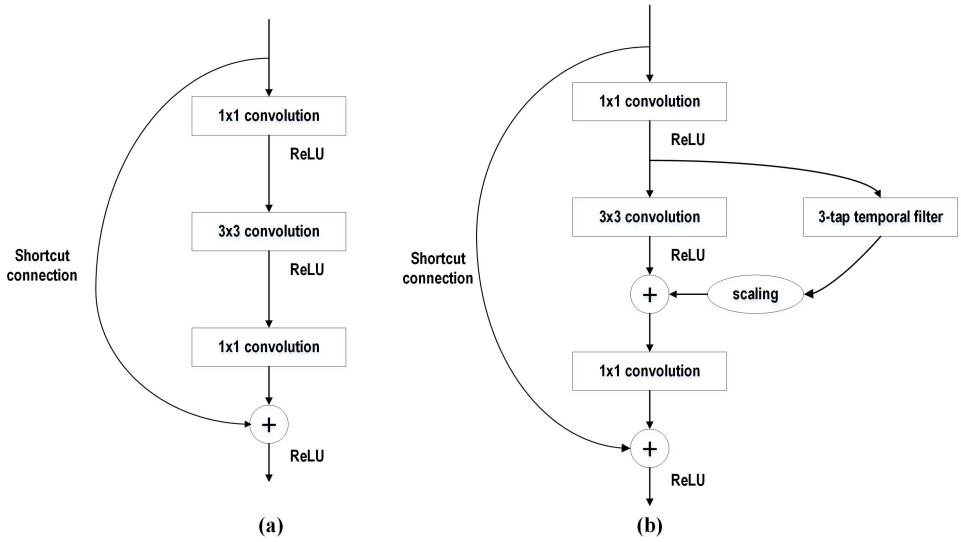


Figure 2.2: The "bottleneck" building block for (a) the original ResNet [64] and (b) the the temporal residual network [52]. The "ReLU" denotes the Rectified Linear Units, and the symbol $\oplus$ denotes the element-wise addition.

Tran *et al*. investigated the role of 3D convolutions in CNN [60, 61]. Their C3D model [60] uses only 3D convolutions and 3D max-pooling operations. Pre-trained on the large-scale Sports 1M video dataset [59], the C3D can be deployed to directly extract a combined spatial and temporal feature from a video segment. In their follow-up research [61], they proposed a R(2+1)D model that decomposes

a 3D spatio-temporal convolution into a 2D spatial convolution followed by a 1D temporal convolution. For example, a 3D filter of size $t \times d \times d$ can be decomposed into 2D filters of size $1 \times d \times d$, followed by a 1D filter of size $t \times 1 \times 1$. This decomposition not only makes the optimization easier, but also adds non-linearity into the network without increasing the number of parameters [61]. The R(2+1)D model was pre-trained on both the Sports 1M dataset and the Kinetics dataset [62]. Carreira and Zisserman's I3D model [62] was obtained by inflating 2D convolutional filters and 2D pooling kernels. Specifically, their model was inspired and based on the Inception V1 module [65]. One issue with 3D CNNs, as pointed out by Simonyan and Zisserman [66], is that a 3D CNN may not be able to capture the temporal information by directly learning from raw video segments. One proof can be found in Tran *et al.*'s own experimental results with their C3D model [60], in which the C3D feature representation did not give an advantage over the spatial feature representation extracted from individual video frames. Besides 3D CNNs, some earlier deep-learning methods were also proposed to learn spatio-temporal features from video segments, *e.g.* the convolutional gated Restricted Boltzmann Machine [67] and independent subspace analysis [68].

2.2.3 Recurrent neural networks

Recurrent neural networks (RNNs) are handy tools to process temporal sequences. In these networks, the output at a previous time step can be used as input at a succeeding time step. In this subsection, several RNNs will be presented, followed with a discussion of Long-Short-Term-Memory (LSTM) networks in video classification. An RNN is illustrated in Figure 2.3. Given a sequence of inputs $\{x_0, x_1, ..., x_{t-1}, x_t, ..., x_N\}$, where $N+1$ is the length of the sequence, the RNN outputs two sequences, $\{y_0, y_1, ..., y_{t-1}, y_t, ..., y_N\}$ and $\{a_0, a_1, ..., a_{t-1}, a_t, ..., a_N\}$. The

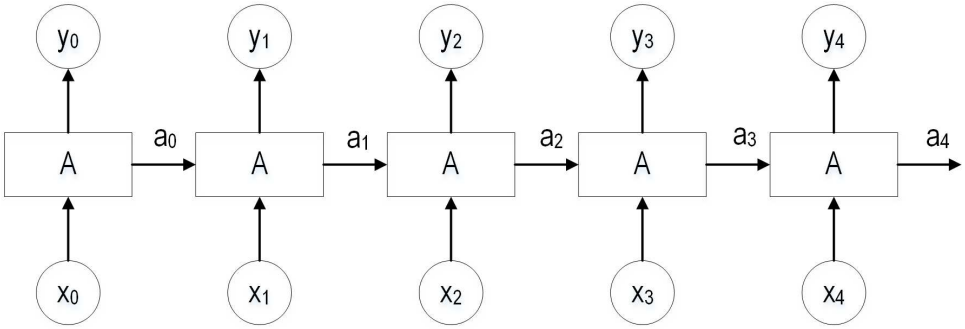


Figure 2.3: An illustration of RNN. It outputs two sequences, $\{y_0, y_1, ..., y_N\}$ and $\{a_0, a_1, ..., a_N\}$ using one sequence $\{x_0, x_1, ..., x_N\}$ as input. In this example $N = 4$. At time step t, both x_t and a_{t-1} are taken to compute a_t.

computations involved are described as follows:

$$\begin{cases} a_t &= \mathrm{f}(x_t, a_{t-1}) \\ y_t &= \mathrm{g}(a_t), \end{cases} \tag{2.23}$$

where f(·) and g(·) are two functions adopted for the computations. The item a_{-1} can be initialized to zero. A vanilla RNN faces the "long-term dependency problem", that is, it is difficult for the RNN to derive information from time steps that are far behind. This is because the gradients of the loss function decay exponentially with time (called the "vanishing gradient problem"). One solution to address this issue is the LSTM network [69].

The core of the LSTM model is a memory cell that controls how much information flows through the network. In turn, the behavior of the memory cell is controlled by three gates—the "forget gate layer", the "input gate layer" and the "output gate layer" (See Figure 2.4). These gates are regulated by the sigmoid function defined as $\sigma(x) = \frac{1}{1+e^{-x}}$. The sigmoid function outputs a number between 0 and 1, with 1 representing "completely keep this" while 0 meaning "completely omit this." Given a sequence $\mathbf{X} = [\mathbf{x}_0, \mathbf{x}_1, ..., \mathbf{x}_t, ..., \mathbf{x}_N]$, where each $\mathbf{x}_t$ is a d-dimensional vector, the following equations delineate the workflow of an LSTM network. Note that here, sequences of vectors are considered instead of 1D time sequences.

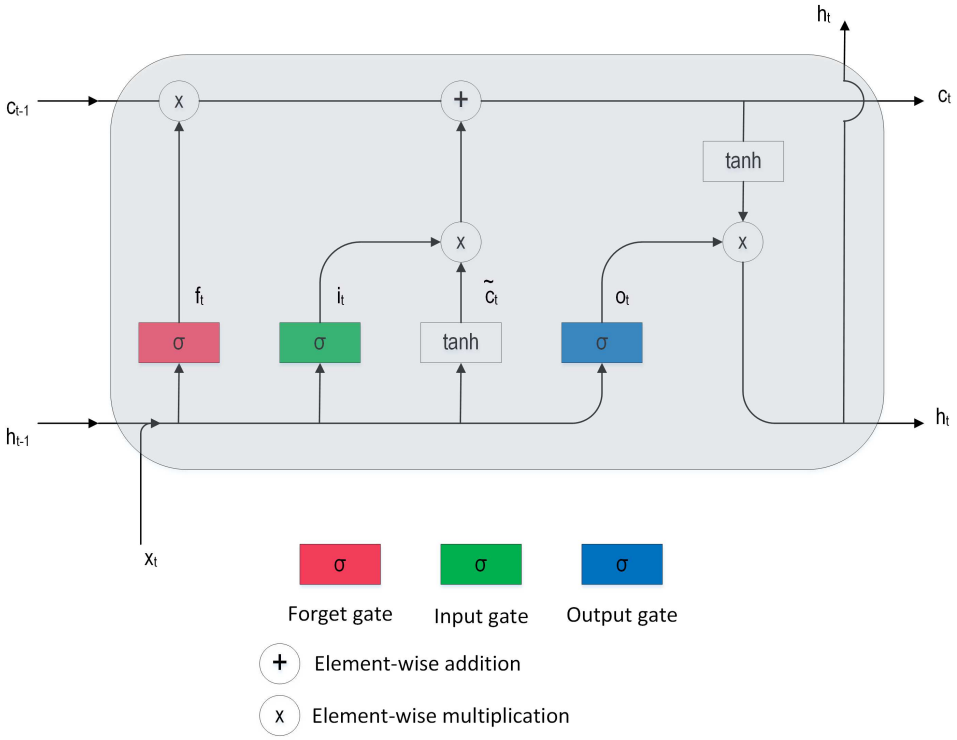


Figure 2.4: An LSTM unit.

$$\begin{cases} \mathbf{f}_t &= \sigma(\mathbf{W}_f \mathbf{x}_t + \mathbf{U}_f \mathbf{h}_{t-1} + \mathbf{b}_f) \\ \mathbf{i}_t &= \sigma(\mathbf{W}_i \mathbf{x}_t + \mathbf{U}_i \mathbf{h}_{t-1} + \mathbf{b}_i) \\ \tilde{\mathbf{c}}_t &= \tanh(\mathbf{W}_c \mathbf{x}_t + \mathbf{U}_c \mathbf{h}_{t-1} + \mathbf{b}_c) \\ \mathbf{c}_t &= (\mathbf{f}_t \otimes \mathbf{c}_{t-1}) \oplus (\mathbf{i}_t \otimes \tilde{\mathbf{c}}_t) \\ \mathbf{o}_t &= \sigma(\mathbf{W}_o \mathbf{x}_t + \mathbf{U}_o \mathbf{h}_{t-1} + \mathbf{b}_o) \\ \mathbf{h}_t &= \mathbf{o}_t \otimes \tanh(\mathbf{c}_t) \end{cases} . \tag{2.24}$$

In the above equations, vector $\mathbf{f}_t$ is the output of the forget gate layer, vector $\mathbf{i}_t$ is the output of the input gate layer, vector $\mathbf{c}_t$ is the cell state and $\tilde{\mathbf{c}}_t$ its candidate value, and $\mathbf{o}_t$ is the output of the output gate layer. The two vectors $\mathbf{h}_{-1}$ and $\mathbf{c}_{-1}$ are initialized to zero. The matrices $\mathbf{W}_f$, $\mathbf{W}_i$, $\mathbf{W}_c$, $\mathbf{W}_o$, $\mathbf{U}_f$, $\mathbf{U}_i$, $\mathbf{U}_c$ and $\mathbf{U}_o$ are weight matrices of appropriate dimensions. The vectors $\mathbf{b}_f$, $\mathbf{b}_i$, $\mathbf{b}_c$ and $\mathbf{b}_o$ are bias vectors.

The LSTM networks have been successfully used for video classification. A commonly-used strategy is to combine a CNN with an LSTM network to se-

quentially process the spatial and temporal information of a video [70, 71]: first, frame-wise spatial features (which describe the static appearance of individual video frames) are extracted using a CNN; then, an LSTM network is used to learn the temporal behavior of these spatial features. The LSTM networks are perhaps the most popular RNNs for sequential data process. A very recent review of the variants of LSTM networks can be found in [72]. Karim *et al*. [73] fed an input time series into two networks. One network is the LSMT network and the other a fully convolutional network. The LSMT network is responsible for analyzing the temporal behavior of the time series, while the fully convolutional network views the time series as a whole volume. The results of the two networks are fused to yield a combined representation of the time series.

Other RNN alternatives to the LSTM networks include the gated recurrent unit (GRU) [74] and bidirectional RNN [75]. GRU uses two gates, the "update gate" and the "reset gate", to address the vanishing gradient problem of a vanilla RNN. A bidirectional RNN is composed of two independent RNNs. The input sequence is fed into one RNN with a forward time order, and into the other RNN with a backward time order. The idea behind this architecture is that future information can be useful in eliminating ambiguity at the current time step. A comprehensive review of recurrent neural networks can be found in [76]. Recently, MacKay *et al*. proposed the reversible RNN to reduce the activation memory cost while retaining the performance of the traditional RNN [77]. Their reversible RNN model may work as a substitute for LSTM networks, which are used as components in the two proposed dynamic scene recognition methods present in **Chapter 4** and **Chapter 5**.

2.2.4 Two-stream CNNs

To disentangle the temporal information from the static appearance in a video segment, two-stream CNNs have been proposed [66, 78, 79, 80]. One stream (called

the spatial stream) captures the static appearance in the individual video frames, while the other stream (called the temporal stream) extracts motion information from the optical flow field. An illustration of a two-stream architecture proposed in [66] is shown in Figure 2.5.

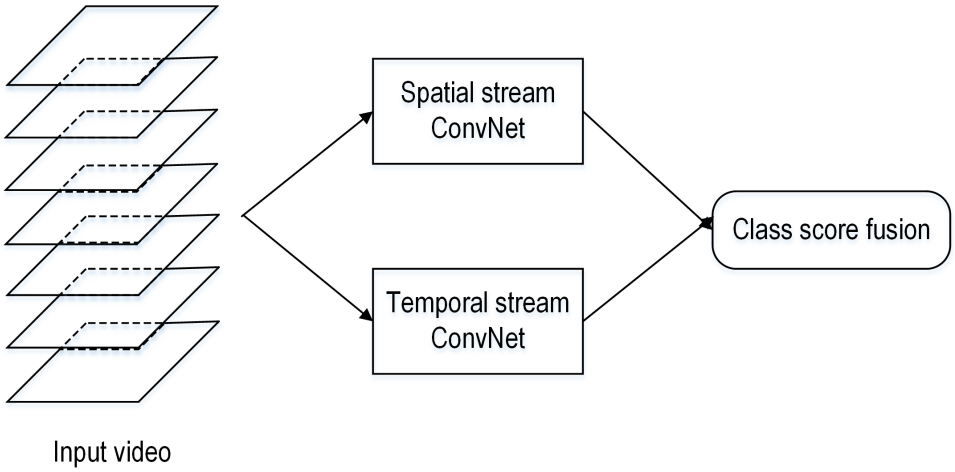


Figure 2.5: Illustration of a two-stream architecture [66]. The input video is fed into two independent CNNs, the outputs of which are fused to obtain the final class score.

The dense optical flow is commonly chosen to form the input to the temporal stream network. Let $\mathbf{d}_t(x,y)$ denote the displacement vector at the pixel location (x,y) in frame t, then the location of the corresponding pixel in frame $t+1$ is $(x,y)+\mathbf{d}_t(x,y)$. The horizontal and vertical components of $\mathbf{d}_t(x,y)$, d_t^{x} and d_t^{y}, can be viewed as two image channels. Thus, for a video segment of length L, an input volume $V \in \mathbb{R}^{w\times h\times 2L}$ can be formed by stacking up these optical flow fields, where w and h are the dimensions of the video frames. Formally,

$$\begin{cases} V(x,y,2t-1) & = d_t^{\mathrm{x}}(x,y) \\ V(x,y,2t) & = d_t^{\mathrm{y}}(x,y) \end{cases}, 1 \le x \le w, 1 \le y \le h, 1 \le t \le L. \tag{2.25}$$

The volume V captures the motion information of the video segment and is hence used as input to the temporal stream. One disadvantage with the original two-stream CNN [66] is that the fusion of the two streams takes place at the class scores, while neglecting the pixel-wise correspondences between the two

streams. Feichtenhofer *et al*. addressed this issue by introducing 3D convolution and 3D pooling to fuse the two streams at the last convolutional layer into the spatial stream, converting the two streams into a spatio-temporal stream [78]. Meanwhile, they also performed 3D pooling in the temporal stream alone. The spatio-temporal stream and the temporal stream are then fused at the class scores. Another disadvantage with the original two-stream CNN is that it can only process short video segments of fixed length. To address this issue, Wang *et al*. [79] proposed the temporal segment networks (TSN). A long video sequence is first divided into multiple segments of equal length. Then, snippets are randomly sampled from each segment and fed into a two-stream CNN. Next, the information of these snippets is aggregated by two "segmental consensus functions" each for a stream. Finally, the results from the two segmental consensus functions are fused at the class score level. Compared to the relatively shallow network architecture [81] of the original two-stream CNN, the authors of [79] introduced the batch normalization (BN) [82] layers into the networks. Lan *et al*. [80] proposed the so called deep local video feature (DOVF) based on the TSN. They used the TSN to extract local features from the short snippets (each snippet has two local features corresponding to the two streams), which are then aggregated into a global feature and classified with SVMs. Li *et al*. [83] proposed an architecture consisting of two convolutional Attention-LSTM networks, with one convolutional Attention-LSTM network processing RGB images and the other processing optical flow images. The more motion significance a region of the video has, the more attention this region receives in the convolutional Attention-LSTM networks.

Wang *et al*.'s temporal pyramid pooling-based convolutional network [84] is an implicit form of two-stream CNN. In this network, the appearance of a frame is represented by the fc7 feature (which are extracted from the fc7 layer of the CNN model), while the motion embedded in the frames is built upon the HoF [21] feature and the MBH [42] feature. The appearance and motion features are concatenated as a frame-level feature. These frame-level features are then

encoded and temporally pooled to yield a video-level feature, which is used for classification. As the motion feature is high-dimensional, the authors used a supervised feature merging method to reduce its dimension [85].

Though two-stream CNNs have proven to outperform a single stream architecture, it is obvious that the performance of the temporal net relies heavily on the computed accuracy of the optical flow field. As the optical flow constraints can be violated in most natural scenes, the performance of two-stream CNNs can be adversely affected in dynamic scene recognition problems. Inspired by two-stream CNNs, Tu *et al*. proposed a multi-stream CNN architecture to incorporate multiple complementary features trained in appearance and motion networks [86].

The performances of deep-learning methods on the two dynamic scene datasets are listed in Table 4.4 and Table 4.5 of **Chapter 4**. By comparing these tables with Table 2.1, one can easily see that deep-learning methods outperform handcrafted features by a significant margin.

2.3 Chapter summary

This chapter comprehensively reviews methods for dynamic scene recognition. The methods discussed and analyzed are roughly divided into two broad categories according to the features they use: handcrafted methods and deep-learning methods. Handcrafted methods traditionally dominated dynamic scene recognition; they also provide insightful solution pipelines. However, deep-learning methods are current state-of-the-art in this field. Some approaches, which are meant for the more broad aim of video classification, are also included in this chapter to provide a wider picture of relevant tools and techniques.

Chapter 3

Review of Mid-level Representations

Chapter contents

The majority of the methods presented in the previous chapter are *global* methods, *i.e.*, they employ global features extracted from the entire video frame or a video segment. However, it has been proven that not all portions of an image are equally important in classification tasks. This chapter reviews *mid-level* feature representations. The mid-level representations lie between the low-level features such as HoG [20] and SIFT [22] features, and the high-level representations that are extracted from the whole image. Three mid-level representations will be presented in this chapter: 1) topic models, 2) part-based models, and 3) bank-based representations. The word "topic" originated from statistical natural language

processing. It is a level higher than "word" but lower than "corpus". The process of topic modeling is to infer hidden variables such as word distributions for all topics and topic mixture distributions for each document by observing a corpus. Part-based models, however, seek those parts of an image that are more representative and discriminative than others. Bank-based representations are based on responses of image patches to a "bank". A bank can be explicitly constructed as an ensemble of filters, or more implicitly appear as a CNN to extract features from image regions. This chapter, along with **Chapter 2**, provides the literature review for **Chapter 4** and **Chapter 5**.

3.1 Topic models

Topic models automatically identify topics present in a text object and derive hidden patterns exhibited by a text corpus. Let us first introduce the related terminology. A *word* is the basic unit defined to be an item from a vocabulary. A *document* is a sequence of words. A *corpus* is a collection of documents. Each document is represented by a mixture of *topics* or *themes*. Hence, topics are an intermediate-level representation. See Figure 3.1 for illustrations.

Now let us relate the scene classification problem with these terms. Fei-Fei and Perona [87] adopted the Latent Dirichlet Allocation (LDA) model [88] to scene classification. In this work, a patch is the basic unit (word) of an image, an image is equivalent to a document, and a category of images constitute a corpus. LDA is a generative probabilistic model of a corpus. The basic idea is that the documents are represented as random mixtures over latent topics, while each topic is characterized by a distribution over words. Consider images of C categories. Each image $\mathcal{X}$ is composed of N patches $\mathcal{X} = \{\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_N\}$. Here a patch is represented by a local feature. The local features extracted from training images are clustered to form a dictionary of T words. With this dictionary, a patch $\mathbf{x}_n$ is represented as a unit vector such that its tth element $x_n^t = 1$ indicates that it

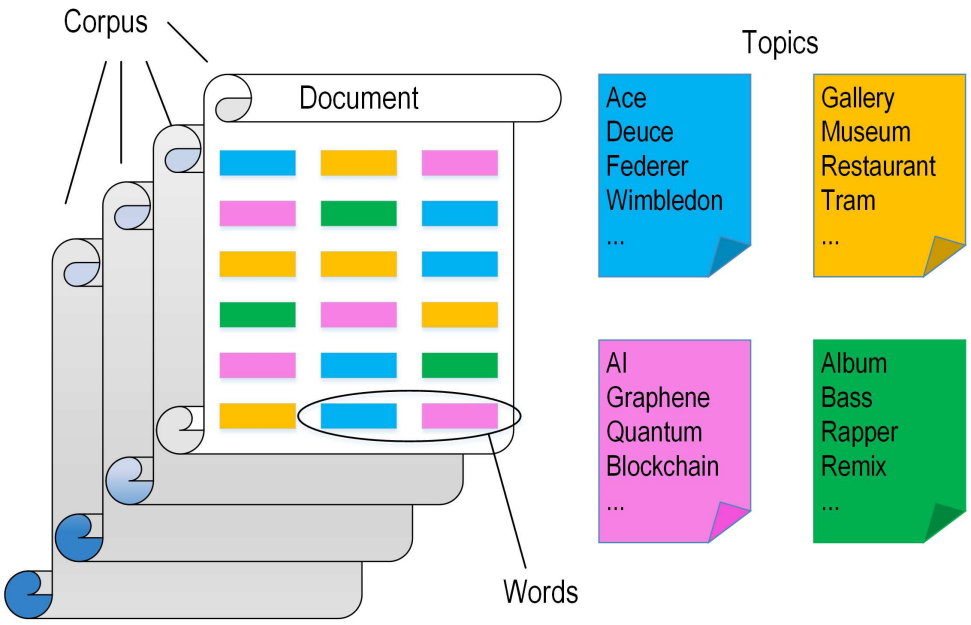


Figure 3.1: An illustration of topic modeling. Here, four topics are established: tennis, city center, computer science and music.

is the tth codeword in the dictionary[a]. A total number of K topics are established. The process of generating an image is summarized in the following steps [87]:

1. Draw a category label c from a uniform distribution $c \backsim p(c) := \mathrm{U}(C)$, where $\mathrm{U}(C)$ is the uniform distribution such that the probability of choosing each category c is $1/C$.

2. For this particular image from category c, draw a parameter $\boldsymbol{\pi}$ (a vector of length K) from a Dirichlet distribution $\boldsymbol{\pi} \backsim \mathrm{Dir}(\boldsymbol{\pi}|\boldsymbol{\theta}_{c\cdot})$, where $\boldsymbol{\theta}$ is a $C \times K$ matrix and $\boldsymbol{\theta}_{c\cdot}$ its cth row. The K-dimensional vector $\boldsymbol{\theta}_{c\cdot}$ is the parameters for the Dirichlet distribution.

3. For each patch $\mathbf{x}_n$ $(1 \leq n \leq N)$ in the image,

   i) Draw a topic $\mathbf{z}_n$ from a multinomial distribution $\mathbf{z}_n \backsim \mathrm{Mult}(\mathbf{z}_n|\boldsymbol{\pi})$, where $\mathbf{z}_n$ is a K-dimensional unit vector such that its kth element $z_n^k = 1$ indicates that the kth topic is selected.

[a]This is obtained by comparing the Euclidean distance between $\mathbf{x}_n$ and the codewords in the dictionary.

ii) Draw a patch $\mathbf{x}_n$ from a distribution $\mathbf{x}_n \backsim p(\mathbf{x}_n|\mathbf{z}_n, \boldsymbol{\beta})$, where $\boldsymbol{\beta}$ is a $K \times T$ matrix whose elements β_{kt} is defined as $\beta_{kt} = p(x_n^t = 1|z_n^k = 1)$.

The above process can be expressed as

$$p(\{\mathbf{x}_n\}_{1\leq n\leq N}, \{\mathbf{z}_n\}_{1\leq n\leq N}, \boldsymbol{\pi}, c|\boldsymbol{\theta}, \boldsymbol{\beta}) = \mathrm{U(C)Dir}(\boldsymbol{\pi}|\boldsymbol{\theta}_{c\cdot}) \prod_{n=1}^{N} \mathrm{Mult}(\mathbf{z}_n|\boldsymbol{\pi}) p(\mathbf{x}_n|\mathbf{z}_n, \boldsymbol{\beta}). \quad (3.1)$$

The multinomial distribution $\mathrm{Mult}(\mathbf{z}_n|\boldsymbol{\pi})$ is expressed as

$$\mathrm{Mult}(\mathbf{z}_n|\boldsymbol{\pi}) = \prod_{k=1}^{K} \pi_k^{z_n^k}, \quad (3.2)$$

where π_k denotes the kth element of $\boldsymbol{\pi}$ and $\sum_{k=1}^{K} \pi_k = 1$. The Dirichlet distribution is a family of prior distributions for the parameter $\boldsymbol{\pi}$ of the multinomial distribution $\mathrm{Mult}(\mathbf{z}_n|\boldsymbol{\pi})$. It is written as

$$\mathrm{Dir}(\boldsymbol{\pi}|\boldsymbol{\alpha}) = \frac{\Gamma(\alpha_0)}{\Gamma(\alpha_1)\ldots\Gamma(\alpha_K)} \prod_{k=1}^{K} \pi_k^{\alpha_k - 1}, \quad (3.3)$$

where $\Gamma(x)$ is the gamma function defined as $\Gamma(x) = \int_0^\infty u^{x-1} e^{-u} du$, and $\alpha_0 = \sum_{k=1}^{K} \alpha_k$. Note here $\boldsymbol{\theta}_{c\cdot}$ has been replaced with $\boldsymbol{\alpha}$ to avoid cluttering. Given an inquiry image $\mathcal{X} = \{\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_N\}$, the posterior probability of it belonging to category c is

$$p(c|\mathcal{X}, \boldsymbol{\theta}, \boldsymbol{\beta}) \propto p(\mathcal{X}|c, \boldsymbol{\theta}, \boldsymbol{\beta}). \quad (3.4)$$

The likelihood $p(\mathcal{X}|c, \boldsymbol{\theta}, \boldsymbol{\beta})$ can be expanded as

$$p(\mathcal{X}|c, \boldsymbol{\theta}, \boldsymbol{\beta}) = \int \mathrm{Dir}(\boldsymbol{\pi}|\boldsymbol{\theta}_{c\cdot}) \left(\prod_{n=1}^{N} \sum_{\mathbf{z}_n} \mathrm{Mult}(\mathbf{z}_n|\boldsymbol{\pi}) p(\mathbf{x}_n|\mathbf{z}_n, \boldsymbol{\beta}) \right) d\boldsymbol{\pi}. \quad (3.5)$$

Eq. (3.5) is not tractable due to the coupling between $\boldsymbol{\pi}$ and $\boldsymbol{\beta}$. However, variational approximation and Markov chain Monte Carlo (MCMC) can be applied to approximately infer this likelihood [88]. The parameters $\boldsymbol{\theta}$ and $\boldsymbol{\beta}$ can be learned

by maximizing the log likelihood $\log p(\mathcal{X}|c, \boldsymbol{\theta}, \boldsymbol{\beta})$ using variational inference [87] or Gibbs sampling [89]. Zeng *et al*. [90] used the loopy belief propagation (LBP) [91] for the approximate inference and parameter estimation for the LDA. Liu *et al*. [92] improved this method by introducing the importance of visual words to the LBP model. The original LDA model is inherently unsupervised. Some researchers developed supervised LDA models [93, 94]. Pan *et al*. replaced the last pooling layer of a conventional CNN with a topic model layer to get a fixed-size output of an image of arbitrary size [95]. This topic model layer is based on the LDA. Recently, Sorkhi *et al*. proposed a comprehensive system for image scene classification, in which topic modeling is used to represent the latent relationships between the objects [96].

Another useful generative model for topic learning is the probabilistic Latent Semantic Analysis (pLSA) [97, 98]. However, compared to pLSA, the LDA model can reduce overfitting [89].

3.2 Part-based models

Since Felzenszwalb *et al*.'s seminal work of deformable part models (DPM) [99], part-based models have drawn increasing attention from researchers. Girshick *et al*. revealed that part-based models have a connection with CNNs [100]. The idea behind part-based models is that not all parts of an image are critical for recognition tasks. Rather, some parts are common and representative for a specific category, while at the same time discriminative with respect to other categories. How to extract representative parts and how to learn discriminative parts are two key respects of these methods. To extract the representative parts from a given image category, a large number of patches are randomly sampled from images of this category and encoded into features, such as HoG and SIFT features. Usually, patches of multiple resolutions are sampled to account for scale changes. The next step is to seek clusters from this large pool of features. Commonly-used

methods for this purpose are *K*-means clustering [101, 102] and spectral clustering [103, 104]. The cluster centers obtained can be used as representative parts. Next, discriminative parts are sought from the representative parts. According to the backbone techniques they use, these methods are grouped into three types: 1) SVM-based methods, 2) "one-vs-all" optimization methods, and 3) "all-in-one" optimization methods. The first group of methods explicitly uses an SVM to "detect" discriminative parts. The latter two groups of methods, however, formulate an optimization problem to learn the discriminative parts; the difference between them is whether some parts are shared or not across different categories.

3.2.1 SVM-based methods

Most methods in this group train linear SVMs to help find discriminative parts. Singh *et al*. trained an SVM to detect discriminative parts after the *K*-means clustering step [101]. To that end, two datasets, the "discovery dataset" $\mathcal{D}$ and the "natural world dataset" $\mathcal{N}$, were constructed. The "discovery dataset" contains images of a specific category from which discriminative parts are to be discovered. The "natural world dataset" contains not only images from this specific category, but also images from other categories. The images in both datasets are divided into two disjoint subsets, *i.e.* $\mathcal{D} = \{\mathcal{D}_1, \mathcal{D}_2\}$ and $\mathcal{N} = \{\mathcal{N}_1, \mathcal{N}_2\}$. Patches are randomly sampled from $\mathcal{D}_1$ and clustered using *K*-means clustering. A linear SVM is trained for each cluster, using the members in this cluster as positive training samples and patches from $\mathcal{N}_1$ as negative training samples. The trained SVMs are then used to detect a small number of supporting members from $\mathcal{D}_2$ to augment their associated clusters. After that, the positions of $\mathcal{D}_1$ and $\mathcal{D}_2$, and $\mathcal{N}_1$ and $\mathcal{N}_2$ are exchanged, to avoid over-fitting caused by working with only one dataset. The above process is repeated until convergence. Finally, the clusters are ranked according to their purity and discriminativeness; the top clusters are chosen as discriminative patches. Doersch *et al*. adopted a similar strategy [105].

However, they used the K nearest-neighbors to preselect candidates for the discriminative parts. In this strategy, a part seeks its K nearest-neighbors in both the intra-category and the inter-category images. The more neighbors are located in the intra-category images, the more likely this part is discriminative. Then, linear SVMs are trained, each for a candidate discriminative part, to further evaluate their discriminativeness. To avoid over-fitting, the training data are divided into several equal sizes. Once the SVMs are trained with a subset of the training dataset, they are used to detect instances in another different subset of the training dataset. In Poselets [106], a linear SVM is trained for each poselet represented by the HoG feature. A poselet is a particular part of the human pose under a given view point, which is in essence a cluster of samples that are close in a 3D configuration space.

Juneja *et al*. proposed a three-step method to seek discriminative parts, called "blocks" [107]. In the "seeding" step, training images are segmented into blocks at multi-scales, and the blocks with immediate sizes are chosen as seeding blocks. The seeding blocks are superpixels; thus, they are more likely to be discriminative than uniformly sampled patches at a regular grid on an image. They are represented by HoG features. In the "expansion" step, each seeding block is formulated as a linear discriminant analysis (LDA) detector to find high-score instances in the intra-category images. This step is conducted in an iterative way such that new blocks similar to the seeding block are iteratively included to train the LDA detector [108]. Finally, in the "selection" step, the seeding blocks are ranked according to their entropy values. Specifically, consider a validation set of C categories. All the seeding blocks' detectors are run on each image in the validation set, and for each image the top r highest scores achieved by all these detectors are recorded. The entropy $H(C|r)$ of a specific detector is defined as

$$H(C|r) = -\sum_{c=1}^{C} p(c|r) \log_2 p(c|r), \tag{3.6}$$

where $p(c|r)$ is the fraction of the r highest scores achieved by this specific detector in the cth category. A lower value of $H(C|r)$ indicates a more discriminative block, as its instances are from a few categories.

Some approaches advocated exemplar-SVMs [109] over conventional linear SVMs. An exemplar-SVM is a special linear SVM. However, different from a conventional linear SVM that classifies a specific category, an exemplar-SVM is learned to classify instances of a specific exemplar. Each exemplar-SVM is thus defined by a single positive instance and millions of negative instances. While an individual exemplar-SVM is quite specific, an ensemble of exemplar-SVMs offers good generalization. Jain *et al*. extended the 2D patches in images to 3D spatio-temporal patches in videos [110]. First, some 3D patches are selected as discriminative candidates. Then, an exemplar-SVM is trained for each candidate. These trained exemplar-SVMs are ranked by two criteria, the "appearance consistency" and the "purity", on a validation set. Endres *et al*. first trained an LDA accelerated version of the exemplar-SVM for a number of part candidates, then selected from these part candidates using the average max precision measure [111]. They further refined the selected part candidates using the appearance consistency and the spatial consistency measures.

One drawback of SVM-based methods is that the number of SVMs for training is linear with the number of candidate discriminative parts. When there are many candidate discriminative parts, the computational burden can be heavy.

3.2.2 One-vs-all methods

Methods of this type train category-specific parts. Intra-category samples are used as positive samples while inter-category samples are used as negative samples. In the pioneering work of DPM [99], a part is represented as a vector $\boldsymbol{\beta}$, and its response $r_{\boldsymbol{\beta}}(\mathbf{x})$ to an image $\mathbf{x}$ is the maximal dot product between $\boldsymbol{\beta}$ and a portion of the image: $r_{\boldsymbol{\beta}}(\mathbf{x}) = \max_z \boldsymbol{\beta}^{\mathrm{T}} \boldsymbol{\phi}(\mathbf{x}, z)$, where z is a latent variable (the position of

the top-left corner of a sub-region in image $\mathbf{x}$), and $\phi(\mathbf{x}, z)$ is the representation of a portion of the image determined by z. Consider a set of N labeled images $\{\mathbf{x}_i, y_i\}_{1 \le i \le N}$, where $y_i = \{-1, 1\}$. The label $y_i = 1$ indicates that the image is a positive sample while $y_i = -1$ indicates otherwise. $\boldsymbol{\beta}$ is learned by solving the following minimization problem:

$$\min_{\boldsymbol{\beta}} \left\{ \frac{1}{2} \|\boldsymbol{\beta}\|_2^2 + \lambda \sum_{i=1}^{N} \max(0, 1 - y_i r_{\boldsymbol{\beta}}(\mathbf{x}_i)) \right\}, \tag{3.7}$$

where λ is a weight to balance the regularization term (the first term of Eq. (3.7)) and the data loss term (the second term of Eq. (3.7)). This minimization problem is solved using a latent SVM [112]. Pandey and Lazebnik used DPM in a weakly supervised fashion [113].

Sun and Ponce [102] optimized all the parts of a specific category simultaneously. In their approach, a part consists of two components, $\boldsymbol{\beta}$ and τ, where $\boldsymbol{\beta}$ is a vector as in the case of DPM, and τ is a scalar bias. Consider a collection of K_c parts for the cth image category, $\{\boldsymbol{\beta}_k, \tau_k\}_{1 \le k \le K_c}$. The response of all these K_c parts generated on an image $\mathbf{x}$ is

$$r_{\{\boldsymbol{\beta}_k, \tau_k\}_{1 \le k \le K_c}}(\mathbf{x}) = \sum_{k=1}^{K_c} \left[\boldsymbol{\beta}_k^{\mathrm{T}} \phi(\mathbf{x}, z_k) - \tau_k \right]_+, \tag{3.8}$$

where z_k is the latent variable defined as $z_k = \arg\max_{z \in \Omega_{\mathbf{x}}} \boldsymbol{\beta}_k^{\mathrm{T}} \phi(\mathbf{x}, z)$, and $\Omega_{\mathbf{x}}$ is the set of all possible part positions in image $\mathbf{x}$. The subscript "+" denotes the operation $[a]_+ = \max(a, 0)$. An optimization problem is formulated as

$$\min_{\{\boldsymbol{\beta}_k, \tau_k\}_{1 \le k \le K_c}, b_c} \left\{ \frac{1}{N} \sum_{i=1}^{N} [1 - y_i (r_{\{\boldsymbol{\beta}_k, \tau_k\}_{1 \le k \le K_c}}(\mathbf{x}_i) + b_c)]_+^2 + \lambda \sum_{k=1}^{K_c} \|\boldsymbol{\beta}_k\|_2 \right\}, \tag{3.9}$$

where b_c is the category-specific bias. The $\ell_{2,1}$ structured sparsity norm $\sum_{k=1}^{K_c} \|\boldsymbol{\beta}_k\|_2$ is introduced to enforce sparsity over the parts, that is, to ensure only a limited number of parts to have strong magnitudes. As the $\ell_{2,1}$ regularization term is non-

smooth, this optimization problem is solved using the FISTA algorithm [114].

Parizi *et al*. modeled a scene with a collection of region models arranged in a reconfigurable manner [115]. An image is divided into a set of pre-defined regions such that what region model is used for what image region is specified by latent variables. Each image region has a preference for some specific region models. For example, for an outdoor scene, regions in the top of the image are formed by a cloud or a sun region model, while regions in the middle and bottom of the image are formed by a tree or a grass region model. The features they used are bag-of-words (BoW) [116] descriptions of SIFT-based visual words.

Li *et al*. [117, 118] combined CNN features with the association rule mining [119]. The association rule mining is a widely-used technique to find associations between many variables. Let $\mathcal{A} = \{a_1, a_2, ..., a_K\}$ denote a set of K items. A *transaction* $\mathcal{T}$ is a subset of $\mathcal{A}$, *i.e.* $\mathcal{T} \subset \mathcal{A}$. Further, let $\mathcal{D} = \{\mathcal{T}_1, \mathcal{T}_2, ..., \mathcal{T}_N\}$ denote a transaction database containing N transactions (N is usually very large). For a given subset $\mathcal{S}$ of $\mathcal{A}$, its *support value* $\text{supp}(\mathcal{S})$ is defined as

$$\text{supp}(\mathcal{S}) = \frac{|\{\mathcal{T} | \mathcal{T} \in \mathcal{D} \wedge \mathcal{S} \subset \mathcal{T}\}|}{N}, \tag{3.10}$$

where $|\cdot|$ denotes the cardinality of a set. Note that $\text{supp}(\mathcal{S})$ is the fraction of the transactions that contain $\mathcal{S}$ and hence $0 \leq \text{supp}(\mathcal{S}) \leq 1$. An *association rule* $\mathcal{S} \rightarrow a$ implies a relationship between $\mathcal{S}$ and item a. For instance, this might imply how likely a customer who bought items in $\mathcal{S}$ would also buy item a. The *confidence* of an association rule is defined as

$$\text{conf}(\mathcal{S} \rightarrow a) = \frac{\text{supp}(\mathcal{S} \cup \{a\})}{\text{supp}(\mathcal{S})}, \tag{3.11}$$

which also ranges between 0 and 1. In the context of image classification, a feature extracted from a layer of a CNN is viewed as a transaction. Here, patches sampled from an image are used as input to the CNN. Each dimension index of

this feature is viewed as an item. At the end of each transaction, a binary label is attached as an extra item. A positive label "+" indicates that the image is from the intra-category, and a negative label "-" denotes otherwise. The association rule mining aims to find subsets of $\mathcal{D}$, denoted $\mathcal{P}$, such that both supp($\mathcal{P}$) and conf($\mathcal{P} \rightarrow$ "+") are large enough. The Aprior algorithm [120] is used to this end. Then, a collection of patches that are both representative and discriminative can be retrieved from $\mathcal{P}$ for this category.

Instead of seeking the modes of a density distribution, Doersch *et al*. used the mean-shift method to seek modes of positive and negative samples density ratio [121]. The data is divided into the positive set and the negative set. Denote the probability densities of these two sets as p_+ and p_-, respectively. Their method locates points in the feature space where the density of the positive set is large while the density of the negative set is small, which is achieved by maximizing the density ratio p_+/p_-.

Sicre and Jurie proposed a part model based on three assumptions [122]: 1) The model is composed of a given number of different parts. 2) Each part of the model is expected to be present in each positive image. 3) Parts should be representatives of the category. Denote a set of training images of C categories as $\mathcal{I} = \mathcal{I}_1 \cup \mathcal{I}_2 \cup ... \mathcal{I}_C$, where $\mathcal{I}_c$ is the set of images for the cth category ($c = 1, 2, ..., C$). For each image I in $\mathcal{I}$, a dense random set of regions is selected, denoted as $\mathcal{R}_I$. A feature is extracted from each region of $\mathcal{R}_I$. Denote $\mathbf{x_r}$ as the feature extracted from region $\mathbf{r} \in \mathcal{R}_I$. The task is to learn a part model consisting of K parts for each category of images. The details are as follows. To impose the second constraint, the authors introduced a *match function* $m(\mathbf{r}, k)$ which is one if region $\mathbf{r}$ is assigned to the kth part and zero otherwise. To impose the first constraint, it is ensured that an image region can be assigned to at most one part, *i.e.* $\sum_{\mathbf{r},k} m(\mathbf{r}, k) \leq 1$. To impose the third constraint, a linear discriminant analysis (LDA) is conducted.

Specifically, each of the K parts is associated with a vector $\mathbf{w}_c(k,m)$ defined as:

$$\mathbf{w}_c(k,m) = \boldsymbol{\Sigma}^{-1}\left[\frac{\sum_{\mathbf{r}\in\mathcal{R}_I, I\in\mathcal{I}_c} m(\mathbf{r},k)\mathbf{x}_\mathbf{r}}{\sum_{\mathbf{r}\in\mathcal{R}_I, I\in\mathcal{I}_c} m(\mathbf{r},k)} - \frac{\sum_{\mathbf{r}\in\mathcal{R}_I, I\in\mathcal{I}} \mathbf{x}_\mathbf{r}}{|\mathbf{r}\in\mathcal{R}_I, I\in\mathcal{I}|}\right], \tag{3.12}$$

where $\boldsymbol{\Sigma}$ is the covariance matrix computed using all the regions in $\mathcal{I}$. The $\mathbf{w}_c(k,m)$'s are learned using an approximation method of combinatorial optimization. Once $\mathbf{w}_c(k,m)$ is available, its response to an image region is computed as $\mathbf{w}_c(k,m)^\mathrm{T}\mathbf{x}_\mathbf{r}$. The global feature representation of an image is obtained as follows: for each part of a model, the region $\mathbf{r}$ that has the highest response is selected and the features of these regions are concatenated into a global feature vector. Their follow-up research [123] adjusted the three assumptions for unsupervised part learning.

One disadvantage of one-vs-all methods lies in their scability—for each image category, an optimization problem needs to be formulated. The all-in-one methods address this issue more efficiently.

3.2.3 All-in-one methods

In this type of methods, parts are shared across different categories or parts from different categories are learned simultaneously. Lobel *et al*. proposed a joint learning framework in which a dictionary and category-specific classifiers are jointly learned [124]. Denote $\mathbf{D} = [\mathbf{d}_1, \mathbf{d}_2, ..., \mathbf{d}_K]$ as a dictionary of K code words, where each column $\mathbf{d}_k$ is a code word. Given an image, a set of features $\{\mathbf{v}_j^l\}$ is extracted from its L distinct regions, where $l = 1,2,...,L$ indexes the L regions and $j = 1,2,...,N_l$ indexes the N_l features from the lth region. Each region is encoded into a K-dimensional vector using *max spatial pooling* described as follows:

$$\mathbf{f}_l = \left[\max_{j=1,2,...,N_l} \mathbf{d}_1^\mathrm{T}\mathbf{v}_j^l, \max_{j=1,2,...,N_l} \mathbf{d}_2^\mathrm{T}\mathbf{v}_j^l, ..., \max_{j=1,2,...,N_l} \mathbf{d}_K^\mathrm{T}\mathbf{v}_j^l\right]^\mathrm{T}, \tag{3.13}$$

Then, the whole image is represented by the concatenation of all these $\mathbf{f}_l$'s, $\mathbf{F} = [\mathbf{f}_1^{\mathrm{T}}, \mathbf{f}_2^{\mathrm{T}}, ..., \mathbf{f}_L^{\mathrm{T}}]$, which has a dimension of KL. Now consider C categories of images. A linear classifier $\mathbf{w}_c \in \mathbb{R}^{KL}$ will be learned for the cth ($c = 1, 2, ..., C$) category such that for any $c' \neq c$, the inequation $\mathbf{w}_c^{\mathrm{T}}\mathbf{F} > \mathbf{w}_c'^{\mathrm{T}}\mathbf{F}$ holds if $\mathbf{F}$ is from the cth category. Given a set of N training images and their category labels $\{\mathbf{F}_i, y_i\}_{i=1,2,...,N}$, $y_i \in \{1, 2, ..., C\}$, the dictionary $\mathbf{D}$ and the classifiers $\mathbf{W} = [\mathbf{w}_1, \mathbf{w}_2, ..., \mathbf{w}_C]$ are jointly learned by formulating the following optimization problem:

$$\min_{\substack{\mathbf{D},\mathbf{W} \\ \{\xi_i\}_{i=1,2,...,N}}} \left\{ \frac{1}{2}\|\mathbf{W}\|_{\mathrm{F}}^2 + \frac{\alpha}{2K}\|\mathbf{D}\|_{\mathrm{F}}^2 + \frac{\beta}{N}\sum_{i=1}^{N}\xi_i \right\}, \tag{3.14}$$

$$\text{s.t. } (\mathbf{w}_c^{\mathrm{T}}\mathbf{F}_i - \mathbf{w}_c'^{\mathrm{T}}\mathbf{F}_i) \geq 1 - \xi_i, \text{ if } y_i = c \text{ and } c' \neq c, \forall i = 1, 2, ..., N,$$

where $\|\cdot\|_{\mathrm{F}}^2$ denotes the squared Frobenius norm, α and β are two weights, $\xi_i \geq 0$ are slack variables. This problem is solved using the Concave-Convex Procedure (CCCP) algorithm [125]. Similar to this method, Parizi *et al*. also trained the part filters (equivalent to the code words in [124]) and the category-specific classifiers jointly [126]. However, different from [124], the classifiers in [126] can take negative values.

Zuo *et al*. proposed the Discriminative and Shareable Feature Learning (DSFL) framework, in which features are shared across different categories [127]. The shareable features are represented by a $K \times d$ matrix $\mathbf{D}$ (its role is similar to that of the dictionary in [124]), called the *feature transformation filter bank*. For each category, a small subset of filters is activated to learn category-specific features. Different categories can share the same filters to learn shareable features. Specifically, for a d-dimensional patch $\mathbf{x}$ represented by a vector of raw pixels, a feature $\mathbf{f} = \mathcal{F}(\mathbf{D}\mathbf{x})$ is obtained that is both discriminative and compact, where $\mathcal{F}(\cdot)$ is an activation function. To that end, the authors of [127] devised an optimization problem comprising three terms: 1) the global reconstruction term, 2) the shareable constraint term, and 3) the discriminative term. The first term is constructed

using all the training samples from all categories, while the last two terms are based on category-specific samples. This optimization problem is solved using an alternating strategy.

Kulkarni *et al*. [128] classified an image by its response to a set of part classifiers, which are shared across all categories. Given an image, a set of features $\mathcal{X} = \{\mathbf{x}_l\}_{l=1,2,...,L}$ is extracted from its L distinct regions, where each $\mathbf{x}_l$ is a d-dimensional vector. The response of this image to a set of P part filters $\boldsymbol{\Theta} = \{\boldsymbol{\theta}_1, \boldsymbol{\theta}_2, ..., \boldsymbol{\theta}_P\}$ is computed. Each part filter $\boldsymbol{\theta}_p$ has K base classifiers. The kth base classifier for the pth part filter has three components, $\{a_{p,k}, \mathbf{u}_{p,k}, b_{p,k}\}$, where $a_{p,k}$ and $b_{p,k}$ are scalars, and $\mathbf{u}_{p,k}$ is a d-dimensional vector. The response of the image to the pth part filter is defined as:

$$f_p(\mathcal{X}) = \sum_{l=1}^{L} \pi_{l,p} \sum_{k=1}^{K} a_{p,k} \sigma(\mathbf{x}_l^{\mathrm{T}} \mathbf{u}_{p,k} + b_{p,k}), \tag{3.15}$$

where $\pi_{l,p} \propto \exp[\beta_p \sum_{k=1}^{K} a_{p,k} \sigma(\mathbf{x}_l^{\mathrm{T}} \mathbf{u}_{p,k} + b_{p,k})]$ and $\sum_{l=1}^{L} \pi_{l,p} = 1$, β_p is a part-dependent "pooling" parameter, and $\sigma(\cdot)$ denotes the sigmoid function. Hence, the responses of the image to all the P part filters are concatenated into a P-dimensional vector $\mathbf{f}(\mathcal{X}; \boldsymbol{\Theta}, \boldsymbol{\beta}) = [f_1(\mathcal{X}), f_2(\mathcal{X}), ..., f_P(\mathcal{X})]^{\mathrm{T}}$, where $\boldsymbol{\beta} = [\beta_1, \beta_2, ..., \beta_P]^{\mathrm{T}}$ is the collection of "pooling" parameters. Further, for C categories of images, a classifier $\mathbf{W} = [\mathbf{w}_1, \mathbf{w}_2, ..., \mathbf{w}_C]$ is also learned, where each $\mathbf{w}_c$ ($c = 1, 2, ..., C$) is a P-dimensional vector. The label y of an image is predicted according to the soft-max probability:

$$\Pr(y = c | \mathcal{X}; \boldsymbol{\Theta}, \boldsymbol{\beta}, \mathbf{W}) = \frac{\exp(\mathbf{w}_c^{\mathrm{T}} \mathbf{f}(\mathcal{X}; \boldsymbol{\Theta}, \boldsymbol{\beta})}{\sum_{c=1}^{C} \exp(\mathbf{w}_c^{\mathrm{T}} \mathbf{f}(\mathcal{X}; \boldsymbol{\Theta}, \boldsymbol{\beta})}. \tag{3.16}$$

Given a set of N training images and their category labels $\{\mathcal{X}_i, y_i\}_{i=1,2,...,N}$, $y_i \in \{1, 2, ..., C\}$, the following optimization is formulated to jointly solve for $\boldsymbol{\Theta}$, $\boldsymbol{\beta}$ and $\mathbf{W}$:

$$\min_{\boldsymbol{\Theta}, \boldsymbol{\beta}, \mathbf{W}} \left\{ -\sum_{i=1}^{N} \sum_{c=1}^{C} [y_i = c] \ln \Pr(y = c | \mathcal{X}; \boldsymbol{\Theta}, \boldsymbol{\beta}, \mathbf{W}) + \mu \|\boldsymbol{\Theta}\|_{\mathrm{F}}^2 + \nu \|\mathbf{W}\|_{\mathrm{F}}^2 \right\}, \tag{3.17}$$

where $[\cdot]$ is the Iverson bracket, and μ and ν are two regularization weights. Note

that the first term of Eq. (3.17) is the cross-entropy loss. This problem is solved with the stochastic gradient descent strategy with respect to the three unknowns.

All-in-one methods are more efficient than one-vs-all methods in terms of formulating less optimization problems. However, the number of training images in these methods is usually quite limited. In the next section, bank-based representations will be discussed that can use a large collection of images.

3.3 Bank-based representations

One type of mid-level representations is based on the responses to a bank. A *bank* is essentially a set of filters. Li *et al*. proposed the object bank image representation [129]. They constructed a bank consisting of 177 frequently appearing objects such as humans, vehicles and water. Given an image, an *object filter* response is generated by running the object filter at various locations and scales of the image in a sliding window manner. Here the filter is an object detector trained on images containing the object of interest [99]. A high response indicates a high possibility of the occurrence of that object at the particular location and scale of the image. Specifically, for each object at each scale, a response map of three spatial levels is constructed, resulting in $n_o \times n_s \times (1^2 + 2^2 + 4^2)$ grids, where n_o is the number of objects and n_s the number of scales. See Figure 3.2 for illustrations.

These responses can be encoded into a global representation for the image using three strategies: the max response representation, the average response representation, and the histogram representation. The action bank [130] is an extension of the object bank to videos. The action filters are the 3D Gaussian third derivative filters of [6].

Rather than using an explicit set of filters to extract the responses from image regions, some methods used CNNs to extract features from the image regions. Wang *et al*. trained a CNN named PatchNet [131], which is mostly based on the Inception V2 architecture [82], but uses patches instead of whole images as input

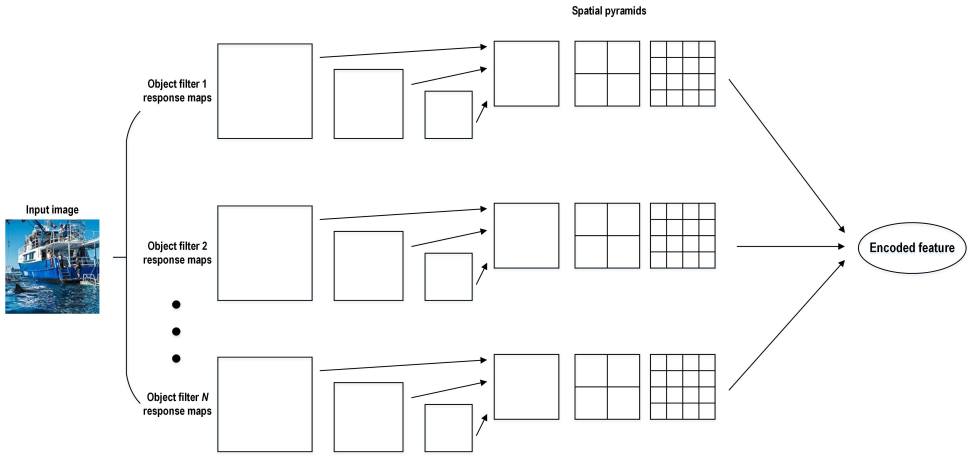


Figure 3.2: An illustration of the object bank image representation [129]. An input image is subjected to a series of object filters at various scales (three scales in this example), resulting in $n_o \times n_s \times (1^2 + 2^2 + 4^2)$ grids for a three-level pyramid. These responses can be encoded into a global representation for the image using three strategies.

to the CNN. The labels of the patches are same as the labels of the images from which they are cropped. The trained PatchNet can then be used to extract features from a given patch. In particular, two features $\mathbf{f}$ and $\mathbf{p}$ are extracted from the patch, where $\mathbf{f}$ is extracted from the last hidden layer of the Inception V2 architecture, while $\mathbf{p}$ is the semantic probability vector. For a total of C categories, feature $\mathbf{p}$ is expressed as $\mathbf{p} = [p^1, p^2, ..., p^C]^{\mathrm{T}}$, where each p^c indicates the probability of this patch belonging to the cth category. Now, given a set of N training patches $\{\mathbf{f}_i, \mathbf{p}_i\}$ $(i = 1, 2, ..., N)$, a semantic codebook of C codewords $\{\boldsymbol{\mu}_c, \boldsymbol{\Sigma}_c\}$ $(c = 1, 2, ..., C)$ is constructed as follows:

$$\begin{aligned} \boldsymbol{\mu}_c &= \frac{1}{N_c} \sum_{i=1}^{N} p_i^c \mathbf{f}_i, \\ N_c &= \sum_{i=1}^{N} p_i^c, \pi_c = \frac{N_c}{N}, \\ \boldsymbol{\Sigma}_c &= \frac{1}{N_c} \sum_{i=1}^{N} p_i^c (\mathbf{f}_i - \boldsymbol{\mu}_c)(\mathbf{f}_i - \boldsymbol{\mu}_c)^{\mathrm{T}}, \end{aligned} \tag{3.18}$$

where p_i^c is the cth element of $\mathbf{p}_i$. The semantic codebook now can be used to aggregate the features extracted from all the regions of an image in a Fisher Vector fashion [15]. Given the features $\{\mathbf{f}_t, \mathbf{p}_t\}$ $(t = 1, 2, ..., T)$ extracted from T local regions

of an image, the first order and second order information of these regions is encoded with respect to the semantic codebook as follows:

$$\mathbf{s}_c = \frac{1}{\sqrt{\pi_c}} \sum_{t=1}^{T} p_t^c \left(\frac{\mathbf{f}_t - \boldsymbol{\mu}_c}{\sigma_c}\right),$$
$$\mathbf{g}_c = \frac{1}{\sqrt{\pi_c}} \sum_{t=1}^{T} p_t^c \left[\frac{(\mathbf{f}_t - \boldsymbol{\mu}_c)^{\mathrm{T}} (\mathbf{f}_t - \boldsymbol{\mu}_c)}{\sigma_c^2} - 1 \right], \tag{3.19}$$

where σ_c is the cth diagonal element of $\boldsymbol{\Sigma}_c$. The vectors $\{\mathbf{s}_c, \mathbf{g}_c\}$ are concatenated into a global feature vector.

Cheng *et al*. [132] built an image representation based on the "objectness" extracted using the pre-trained ImageNet [54]. Given a set of N patches $\mathcal{P} = \{\mathbf{p}_1, \mathbf{p}_2, ..., \mathbf{p}_N\}$ from an image I, features are extracted from the soft-max layer of the pre-trained ImageNet, resulting in object score vectors $\mathcal{S} = \{\mathbf{s}_1, \mathbf{s}_2, ..., \mathbf{s}_N\}$. Each $\mathbf{s}_i$ ($i = 1, 2, ..., N$) is a 1000-dimensional vector; each of its element indicates the probability of patch $\mathbf{p}_i$ belonging to one of 1,000 pre-defined objects. The objectness of the image with respect to the oth ($o = 1, 2, ..., 1000$) object is defined as:

$$\text{objectness}_o(I) = \sum_{\mathcal{S}} s_i^o, \tag{3.20}$$

where s_i^o is the oth element of $\mathbf{s}_i$. Consider C categories of images and denote the set of training images for the cth ($c = 1, 2, ..., C$) category as $\mathcal{I}_c$. Then, the category-level likelihood for the oth pre-defined object is defined as:

$$p(o|c) = \frac{1}{|\mathcal{I}_c|} \sum_{I \in \mathcal{I}_c} \text{objectness}_o(I). \tag{3.21}$$

The authors of [132] sought discriminative objects that have a high appearance probability in a given category. Those parts of an image that have non-discriminative objects will be excluded from representing the image. Finally, an image is represented by the VLAD feature [48] built using only those parts that have discriminative objects. Similarly, Zhu also represented an image us-

ing the salient objects detected in the image [133]. However, the author used a fully-connected layer to fuse the deep features of the salient objects.

One issue with the methods of [131, 132, 133] is that the object types in the pre-trained CNNs are pre-defined, which may not be suitable to problems that have very different object types. Further, it may not be justifiable for the labels of an image's patches to be the same as the image label in [131]. For example, a beach scene image may contain several objects such as ships, humans, sea and sands. Requiring all the objects to share a same label will cause confusion to the CNN. This drawback can be partly alleviated by applying a clustering procedure to the patches instead of assigning a hard label to them [134]. By contrast, the object types are refined in the proposed part-based dynamic scene recognition approach present in **Chapter 4**.

Finally, the performances of some mid-level representation methods are listed in Table 3.1 to enable a comparison between them. The statistics in the table are copied from the corresponding references that reported the results obtained by these methods on four datasets: Dynamic Data Subset [92], 15-Scenes [135], MIT-indoor [136], and UIUC-Sports [129]. Though these methods were not tested on all the four datasets, Table 3.1 still reveals useful insights into these methods. Among them, topic models perform the worst, achieving a noticeable lower accuracy than the other two types of methods on 15-Scenes. Part-based models are generally outperformed by bank-based representations (except for the Object Bank [129]).

3.4 Chapter summary

This chapter comprehensively reviews mid-level representations. The methods discussed and analyzed are roughly divided into three categories: 1) topic models, 2) part-based models, and 3) bank-based representations. Compared with high-level features directly extracted from a whole image or a video, mid-level representations offer a useful alternative for image or video representation.

Table 3.1: Recognition accuracy results (in %) obtained by some mid-level representation methods on four datasets.

	Topic models			**Part-based models**				**Bank-based representations**		
Dataset	GS-LDA [87]	pLSA [98]	KTMBP [92]	BoP [107]	DP [102]	DSFL [127]	SPLeaP [128]	OB [129]	PatchNet [131]	Objectness [132]
Dynamic Data Subset	88.0	87.0	89.0	-	-	-	-	-	-	-
15-Scenes	-	-	72.6	-	87.2	84.2	-	80.9	-	95.9
MIT-indoor	-	-	-	43.6	58.1	52.2	73.5	37.6	84.9	86.8
UIUC-Sports	-	-	-	-	86.8	86.5	-	76.3	-	-

Chapter 4

A Part-Based Method for Dynamic Scene Recognition

Chapter contents

This chapter is based on the following publications:

1. X. Peng, A. Bouzerdoum, "Part-based feature aggregation method for dynamic scene recognition," The International Conference on Digital Image

Computing: Techniques and Applications (DICTA), Perth, Australia, 2019, pp. 1–8.

2. X. Peng, A. Bouzerdoum, S. L. Phung, "A part-based spatial and temporal aggregation method for dynamic scene recognition," *Under revision*, invited submission to Neural Computing and Applications Special Issue on DICTA 2019.

Existing methods for dynamic scene recognition mostly use global features extracted from the entire video frame or a video segment. In this chapter, a part-based method is proposed to aggregate local features from video frames. A pre-trained Fast R-CNN model is used to extract local convolutional features from the regions of interest (ROIs) of training images. These features are clustered to locate representative parts. A set cover problem (SCP) is then formulated to select the discriminative parts, which are further refined by fine-tuning the Fast R-CNN model. Local features from a video segment are extracted at different layers of the fine-tuned Fast R-CNN model, aggregated separately, and then concatenated into a global feature representation. Experimental results show that the proposed method outperforms seven state-of-the-art features on two benchmark dynamic scene datasets.

4.1 Introduction

The task of scene *recognition* (also called *categorization* or *classification*) aims to recognize the semantic label of a given scene, *e.g.*, a beach, an indoor environment, or a city street. Significant effort has been devoted to scene recognition using static images [136, 137, 138]. However, the focus of this chapter is on dynamic scene recognition [6, 7, 8, 9, 10, 11, 31, 49, 50, 51, 52, 139]. Here, dynamic scene recognition is distinguished from dynamic texture recognition [27] and human activity recognition. Dynamic scenes differ from dynamic textures in two respects [49]. First, dynamic scenes are natural scenes evolving overtime, whereas dynamic

textures contain richer texture information. Second, videos of dynamic scenes may contain significant camera movements, while videos of dynamic textures are usually more stable. Dynamic scene recognition also differs from human activity recognition in that the aim of the latter is to recognize the semantic label of a particular human activity happening in a scene, *e.g.*, a sporting activity. Interestingly, video-based dynamic scene classification was first introduced in the context of human action recognition, which proved that scene context information can help improve action recognition [5]. A human activity recognition scenario typically contains artificial objects and humans, which exhibit sharp boundaries in the captured image. These sharp boundaries facilitate the application of the optical flow constraints in tracking trajectories [41] or modeling dense optical flow [66], which are critical to describing the dynamics of a video. However, unlike a human activity recognition scenario that typically contains a significant number of artificial objects and humans, a dynamic scene may only contain highly non-rigid objects with very irregular movements. An example is shown in Figure 4.1. Thus, human activity recognition methods that rely on the optical flow constraints are not sufficient for dynamic scene recognition problems.

(a) **(b)**

Figure 4.1: Comparison of (a) a human activity recognition scene (shooting bow) and (b) a natural dynamic scene (avalanche). The objects and humans in (a) can exhibit sharp boundaries that imply optical flow constraints. By contrast, the optical flow constraints can be violated in most natural scenes such as (b).

Most dynamic scene recognition methods employ *global* features extracted from the entire video frame or a video segment [6, 7, 8, 10, 11, 31, 49, 50, 51, 52,

60]. By contrast, inspired by the success of mid-level representations in image classification [99, 101, 102, 105, 106, 107, 111, 118, 126, 127, 128], here a part-based framework is proposed that aggregates local features within a video segment. It comprises three main novelties. First, after representative parts are automatically learned from a training set, an SCP is formulated to select the discriminative parts (Section 4.2.2); to the best of our knowledge, this is the first time SCP is used for this purpose. Second, refining the selected parts is realized by fine-tuning a Fast R-CNN model [140] instead of designing a complex optimization scheme (Section 4.2.3). Fast R-CNN was originally proposed for object detection that usually requires manually annotated object category labels and bounding boxes. Here part instances are viewed as objects and annotated automatically. Third, a new technique is proposed to temporally aggregate local fully-connected layer features within a video segment (Section 4.2.4.2).

This chapter is organized as follows. Section 4.2 presents the proposed part-based dynamic scene recognition method. Experimental results along with an extensive comparison with seven state-of-the-art methods on two dynamic scene datasets are presented in Section 4.3. Finally, Section 4.4 summarizes this chapter. This chapter does away with the "related work" section, as comprehensive reviews of the related work can be found in **Chapter 2** and **Chapter 3**.

4.2 Proposed part-based dynamic scene recognition method

A part-based dynamic scene recognition method is proposed that consists of six steps: 1) feature initialization, 2) feature clustering, 3) part selection, 4) part refinement, 5) spatial and temporal information aggregation, and 6) classification. In Step 1, a pre-trained Fast R-CNN model is deployed to extract conv5_3 features from a large number of ROIs in training images. The conv5_3 feature corresponds

to the "conv5_3" layer of the VGG-16 architecture [56]. They contain abundant information about local regions of an image [50]. In Step 2, the extracted features are clustered for each category separately. This step allows us to locate the representative features for each category, which are called "representative parts" in this chapter. Then, in Step 3, discriminative parts are selected from the representative parts by formulating an SCP optimization problem. In Step 4, the discriminative parts are further refined. This is obtained by fine-tuning the Fast R-CNN model, which is then used to extract local features from the ROIs. In Step 5, the extracted local features are aggregated to form feature representations, both spatially and temporally, for a video segment. Finally, in Step 6, the video-segment-level features are used for classification. The entire pipeline of the proposed method is illustrated in Figure 4.2 with details following in the text.

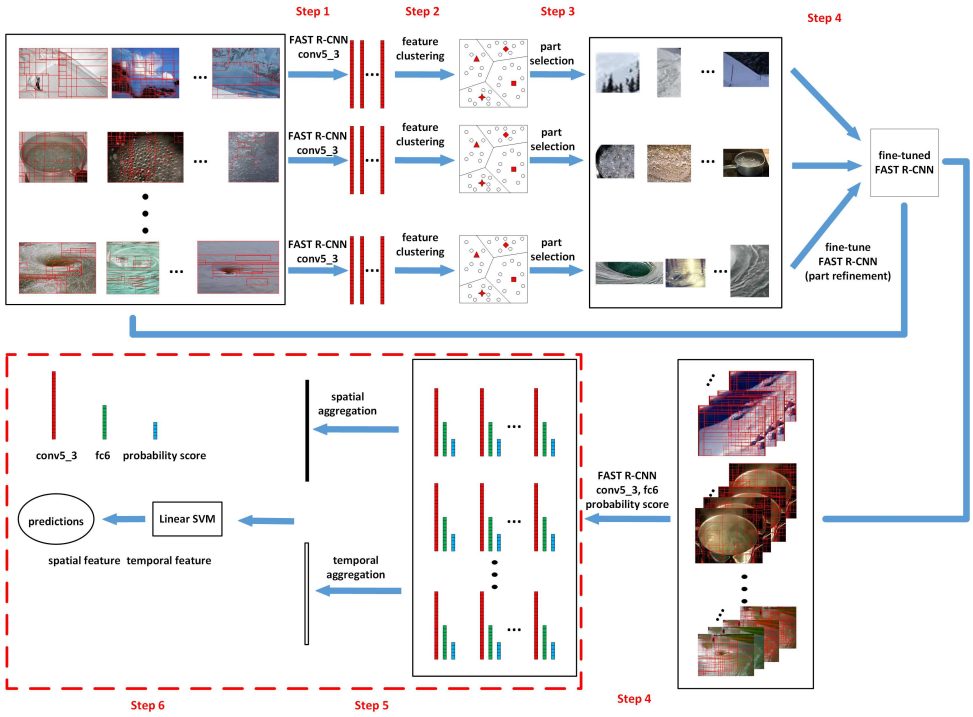


Figure 4.2: The pipeline of the proposed part-based dynamic scene recognition method. It consists of six steps. Red rectangles inside images denote regions of interest (ROIs). Steps 5 and 6 are shared by both the training and testing stage (indicated in the red dash line box).

4.2.1 Representative part extraction

In this step, a set of ROIs is first collected from training images. Then, the collected ROIs are fed into a pre-trained fast R-CNN to extract conv5_3 features, which are clustered for each scene category. The obtained cluster medoids constitute the representative parts of each category. Consider a dataset $\mathcal{I}$ of training images for C scene categories. First, a set of ROIs is collected from the training images using the selective search technique [141], which is a fast method to generate region proposals that may contain objects of interest in an image. Next, a pre-trained Fast R-CNN is deployed to extract conv5_3 features from the collection of ROIs. The conv5_3 features of each category are clustered into K clusters, see Algorithm 1. Note that Algorithm 1 differs from the conventional K-means clustering algorithm in Step (iii), where cluster medoids $\mathbf{v}'$ instead of cluster centroids $\overline{\mathbf{y}}_{c,k}$ are chosen. This is because the conv5_3 features in $\mathcal{F}_c$ are highly sparse and the mean operation destroys the sparsity. However, Algorithm 1 is not a conventional K-medoids algorithm either; in our case, a cluster medoid is computed based on its distance to a cluster centroid rather than to the other cluster members. Algorithm 1 is faster than the popular Partitioning Around Medoids (PAM) algorithm for K-medoid clustering. Let $\mathcal{R}_c = \{\mathbf{r}_{c,1}, \mathbf{r}_{c,2}, ..., \mathbf{r}_{c,K}\}$ denote the K cluster medoids of category c; they are used as the representative parts. For each $\mathbf{r}_{c,k}$, its Euclidean distances to all the members in the corresponding cluster $\mathcal{Y}_{c,k}$ are computed and ranked. Two thresholds to be used in later steps, $d^{10}_{c,k}$ and $d^{25}_{c,k}$ are calculated. The value $d^{10}_{c,k}$ regulates that only 10% of the ranked distance values are smaller than it; the value $d^{25}_{c,k}$ controls that only 25% of the ranked distance values are smaller than it.

4.2.2 Part selection

Having identified the K representative parts of each category, the next step is to determine the most discriminative parts. Here an SCP is formulated for that

Algorithm 1 Clustering procedure for obtaining representative parts

Input: $\mathcal{F}_c$ – set of conv5_3 features for scene category c ($c = 1, 2, ..., C$)
K – preselected number of clusters
Output: $\mathcal{R}_c = \{\mathbf{r}_{c,1}, \mathbf{r}_{c,2}, ..., \mathbf{r}_{c,K}\}$ – set of K cluster medoids of category c, which are used as the representative parts of this category
Steps: (i) Randomly initialize $\mathcal{R}_c$ from $\mathcal{F}_c$. For each $\mathbf{r}_{c,k} \in \mathcal{R}_c$, initialize an empty set $\mathcal{Y}_{c,k} = \emptyset$.
(ii) Add each feature $\mathbf{f}_i$ in $\mathcal{F}_c$ into set $\mathcal{Y}_{c,k'}$, where $k' = \arg\min_k \|\mathbf{r}_{c,k} - \mathbf{f}_i\|_2$.
(iii) Compute the centroids $\overline{\mathbf{y}}_{c,k}$ of each set $\mathcal{Y}_{c,k}$, and find $\mathbf{v}'$ in $\mathcal{Y}_{c,k}$ such that $\mathbf{v}' = \arg\min_{\mathbf{v} \in \mathcal{Y}_{c,k}} \|\mathbf{v} - \overline{\mathbf{y}}_{c,k}\|_2$. Replace $\mathbf{r}_{c,k}$ with $\mathbf{v}'$.
(iv) Reinitialize $\mathcal{Y}_{c,k}$ to an empty set. Repeat steps (ii) and (iii) until convergence.

purpose. The SCP formulation offers an elegant way to choose the minimal number of representative parts that are complementary to each other in terms of discriminativeness. To this end, two empty sets, $\mathcal{G}_{c,k}^{\text{intra}}$ and $\mathcal{G}_{c,k}^{\text{inter}}$, are initialized for each representative part $\mathbf{r}_{c,k}$. They will be expanded to contain the instances of $\mathbf{r}_{c,k}$ in the intra-category case and the inter-category case as explained below. Given an image $I \in \mathcal{I}$ and its set of conv5_3 features extracted from its ROIs, $\mathcal{F}(I)$, an instance of $\mathbf{r}_{c,k}$ is found in the image I if any $\mathbf{f} \in \mathcal{F}(I)$ satisfies $\|\mathbf{r}_{c,k} - \mathbf{f}\|_2 \leq d_{c,k}^{25}$. If the image I is from category c, the set $\mathcal{G}_{c,k}^{\text{intra}}$ will be expanded; otherwise, the set $\mathcal{G}_{c,k}^{\text{inter}}$ will be expanded. The more discriminative is $\mathbf{r}_{c,k}$, the more its instances will be found in the intra-category case than in the inter-category case; therefore, the size of $\mathcal{G}_{c,k}^{\text{intra}}$ will be larger than that of $\mathcal{G}_{c,k}^{\text{inter}}$. Algorithm 2 is applied to expand $\mathcal{G}_{c,k}^{\text{intra}}$ and $\mathcal{G}_{c,k}^{\text{inter}}$, where $l(I)$ denotes the category label of the image $I \in \mathcal{I}$. An SCP is formulated to select the discriminative parts based on $\mathcal{G}_{c,k}^{\text{intra}}$ and $\mathcal{G}_{c,k}^{\text{inter}}$.

Algorithm 2 Procedure for expanding $\mathcal{G}_{c,k}^{\text{intra}}$ and $\mathcal{G}_{c,k}^{\text{inter}}$

for each $\mathbf{r}_{c,k}$ and each $I \in \mathcal{I}$ **do**
  if any $\mathbf{f} \in \mathcal{F}(I)$ satisfies $\|\mathbf{r}_{c,k} - \mathbf{f}\|_2 \leq d_{c,k}^{25}$ **then**
    Add I into $\mathcal{G}_{c,k}^{\text{intra}}$, if $l(I) = c$ and $I \notin \mathcal{G}_{c,k}^{\text{intra}}$.
    Add I into $\mathcal{G}_{c,k}^{\text{inter}}$, if $l(I) \neq c$ and $I \notin \mathcal{G}_{c,k}^{\text{inter}}$.
  end if
end for

The general SCP: Consider a set of N elements $\mathcal{T} = \{t_1, t_2, ..., t_N\}$ and a collection

of K subsets, $\mathcal{S}_1, \mathcal{S}_2, ..., \mathcal{S}_K \subset \mathcal{T}$; each subset $\mathcal{S}_k$ is associated with a weight w_k. The SCP is formulated as

$$\min \sum_{k \in \mathcal{P}} w_k, \quad \text{s.t.} \bigcup_{k \in \mathcal{P}} \mathcal{S}_k = \mathcal{T}, \tag{4.1}$$

where set $\mathcal{P} \subseteq \{1, 2, ..., K\}$.

Concerning part selection, the set $\mathcal{T}$ in Eq. (4.1) consists of the images in $\mathcal{I}$ that are from category c, the subset $\mathcal{S}_k$ is $\mathcal{G}_{c,k}^{\text{intra}}$, and the weight w_k is the size of $\mathcal{G}_{c,k}^{\text{inter}}$. As explained before, a smaller size of $\mathcal{G}_{c,k}^{\text{inter}}$ indicates a more discriminative part; therefore, by minimizing $\sum_{k \in \mathcal{P}} w_k$ the discriminative parts tend to be selected. Meanwhile, the constraints in Eq. (4.1) intend to retain those parts that are more representative, as it is desirable that $\bigcup_{k \in \mathcal{P}} \mathcal{S}_k$ can cover $\mathcal{T}$ (which means the selected parts should have instances in all the intra-category images).

The SCP is inefficient to solve with an exact method because it is NP-complete. An approximation method is adopted to solve this problem [142]. Specifically, a dual linear program (LP) problem is formulated as

$$\max \sum_{n \in \{1,2,...,N\}} y_n, \quad \text{s.t.} \begin{cases} \sum_{t_n \in \mathcal{S}_k} y_n \leq w_k \\ y_n \geq 0, n \in \{1, 2, ..., N\} \end{cases}. \tag{4.2}$$

After the optimal solution y^* to the above dual LP problem is found, the set $\mathcal{P}$ in Eq. (4.1) is constructed as follows:

(i) Initialize $\mathcal{P}$ to an empty set.

(ii) If $\mathcal{S}_k$ satisfies $\sum_{t_n \in \mathcal{S}_k} y_n^* = w_k$, add k into $\mathcal{P}$, which means the corresponding part $\mathbf{r}_{c,k}$ is selected. Now, a collection of representative and discriminative parts $\{\mathbf{r}_{c,k}\}_{1 \leq k \leq K_c}$ is obtained for each category c, where K_c is the number of selected parts for category c, which will be further refined.

4.2.3 Part refinement

The parts selected in the previous step need to be refined. This is because the conv5_3 features used until now were extracted using a pre-trained Fast R-CNN model. The pre-defined object types in this model, such as dogs and televisions, are not suitable for other problems. The part refinement is obtained by fine-tuning the Fast R-CNN model. As will be shown shortly, it also exempts us from formulating a complex optimization problem [102, 127, 128]; meanwhile, parts can be shared easily across different categories. For part $\mathbf{r}_{c,k}$ and its two distance thresholds $d^{10}_{c,k}$ and $d^{25}_{c,k}$, the following technique is proposed to provide annotations required for the fine-tuning. For an image $I \in \mathcal{I}$ and its conv5_3 feature set $\mathcal{F}(I)$, part $\mathbf{r}_{c,k}$ is compared against all the features $\mathbf{f} \in \mathcal{F}(I)$. Two conditions are checked:

$$\begin{cases} \text{Condition 1}: & \|\mathbf{r}_{c,k} - \mathbf{f}\|_2 \leq d^{25}_{c,k} \text{ and } c = l(\mathbf{I}). \\ \text{Condition 2}: & \|\mathbf{r}_{c,k} - \mathbf{f}\|_2 \leq d^{10}_{c,k} \text{ and } c \neq l(\mathbf{I}). \end{cases} \tag{4.3}$$

If either condition is satisfied, the ROI of feature $\mathbf{f}$ is annotated as an object bounding box and the index of $\mathbf{r}_{c,k}$ as the "object" category label. The total number of "objects" for the fine-tuned model is $Q = \sum_{c=1}^{C} K_c$, which is the total number of selected parts for all the C categories. Condition 1 deals with part detection for the intra-category case, whereas Condition 2 is used to share parts across categories by allowing $\mathbf{r}_{c,k}$ to be detected in an image from another category. Because $d^{10}_{c,k} < d^{25}_{c,k}$, Condition 2 is stricter than Condition 1 to avoid over-sharing of parts across categories. It is worth pointing out that the parts are refined *implicitly* in our method rather than explicitly as in [102, 127, 128]—the fine-tuned Fast R-CNN model does not directly yield the refined parts; instead, they are embedded in the Fast R-CNN model as refined "object" detectors.

4.2.4 Feature aggregation

The fine-tuned model is now ready to be applied to video data. Consider a video segment of L frames. For each video frame it extracts three features from each of its ROIs: (a) the conv5_3 feature (25,088-dimensional, denoted $\mathbf{u}$), (b) the fc6 feature from the first fully connected layer of the VGG-16 model (4096-dimensional, denoted $\mathbf{v}$), and (c) the probability score vector $\mathbf{p} = [p_1, p_2, ..., p_q, ..., p_Q, p_\mathrm{b}]^T$, where p_q is the probability of an ROI belonging to the qth part, and p_b the probability for the ROI to belong to the background. Note that the conv5_3 feature and the fc6 feature are complementary features. For a given ROI, if $p_\mathrm{b} < 0.5$, it is considered a *non-background* ROI; its conv5_3 feature $\mathbf{u}$ is added to a set $\mathcal{U} = \{\mathbf{u}_1, \mathbf{u}_2, ..., \mathbf{u}_N\}$, and its fc6 feature $\mathbf{v}$ is added to another set $\mathcal{V} = \{\mathbf{v}_1, \mathbf{v}_2, ..., \mathbf{v}_N\}$, where N is the number of non-background ROIs within the video segment. The features in $\mathcal{U}$ and $\mathcal{V}$ will be aggregated separately.

4.2.4.1 Spatial aggregation of $\mathcal{U}$

The features in $\mathcal{U}$ are aggregated using the IFV representation [15]. Let $\Theta = \{\boldsymbol{\mu}_m, \boldsymbol{\Sigma}_m, \pi_m\}_{1 \le m \le M}$ be the parameters of a Gaussian Mixture Model (GMM) fitting the distribution of a large collection of feature vectors, where $\boldsymbol{\mu}_m$, $\boldsymbol{\Sigma}_m$, and π_m denote respectively the mean vector, diagonalized covariance matrix, and weight of the mth GMM component, and M is the total number of GMM clusters. Consider a set of Z-dimensional feature vectors $\mathcal{B} = \{\mathbf{b}_1, \mathbf{b}_2, ..., \mathbf{b}_N\}$, where N is the total number of feature vectors. For a vector $\mathbf{b}_n$ and the mth GMM component, a weight $\omega_{n,m}$ is assigned as

$$\omega_{n,m} = \frac{\exp(-\frac{1}{2} d_{n,m})}{\sum_{m=1}^{M} \exp(-\frac{1}{2} d_{n,m})}, \tag{4.4}$$

where $d_{n,m} = (\mathbf{b}_n - \boldsymbol{\mu}_m)^\top \boldsymbol{\Sigma}_m^{-1} (\mathbf{b}_n - \boldsymbol{\mu}_m)$. Let $b_{n,z}$, $\mu_{m,z}$, and $\sigma_{m,z}$ denote the zth element of $\mathbf{b}_n$, $\boldsymbol{\mu}_m$, and $\boldsymbol{\Sigma}_m$, respectively. For the mth GMM component, two vectors $\boldsymbol{\alpha}_m$ and $\boldsymbol{\beta}_m$ are computed as follows, where $\alpha_{m,z}$ and $\beta_{m,z}$ are the zth element of $\boldsymbol{\alpha}_m$

and $\boldsymbol{\beta}_m$.

$$\begin{cases} \alpha_{m,z} &= \frac{1}{N\sqrt{\pi_m}} \sum_{n=1}^{N} \omega_{n,m} (\frac{b_{n,z}-\mu_{m,z}}{\sigma_{m,z}}) \\ \beta_{m,z} &= \frac{1}{N\sqrt{\pi_m}} \sum_{n=1}^{N} \omega_{n,m} [(\frac{b_{n,z}-\mu_{m,z}}{\sigma_{m,z}})^2 - 1] \end{cases}. \tag{4.5}$$

The IFV representation for the feature set $\mathcal{B}$ is the concatenation of all the $\boldsymbol{\alpha}_m$'s and $\boldsymbol{\beta}_m$'s, $\boldsymbol{\phi}_{\mathrm{s}} = [\boldsymbol{\alpha}_1, \boldsymbol{\alpha}_2, ..., \boldsymbol{\alpha}_M, \boldsymbol{\beta}_1, \boldsymbol{\beta}_2, ..., \boldsymbol{\beta}_M]^\top$, which has $2MZ$ dimensions. To generate an IFV of a smaller dimension, each $\mathbf{u}$ in $\mathcal{U}$ is reshaped into a 512×49 matrix, where each column of the matrix represents a new feature vector of 512 dimensions. Consequently, set $\mathcal{U}$ consists of $49N$ feature vectors of 512 dimensions. Further, the principal component analysis (PCA) is applied for dimension reduction, followed with a GMM fitting. The IFV representation for $\mathcal{U}$ is denoted by $\boldsymbol{\phi}_{\mathrm{s}}$, where the subscript "s" indicates "spatial".

4.2.4.2 Temporal aggregation of $\mathcal{V}$

To temporally aggregate the features in $\mathcal{V}$, they are grouped both *frame-wise* and *part-wise*: first, the fc6 features from frame l ($1 \leq l \leq L$) are placed in a group $\mathcal{H}_l$. Then, each $\mathbf{v} \in \mathcal{H}_l$ is assigned to a subgroup $\mathcal{H}_{l,q} \subset \mathcal{H}_l$, if $p_q > p_{q'}$ for all $q' \neq q$. The following method is proposed to aggregate the temporal information based on the subgroups $\{\mathcal{H}_{l,q}\}$.

First, members in each subgroup $\mathcal{H}_{l,q}$ are sum-pooled:

$$\mathbf{h}_{l,q} = \frac{1}{N_{\mathcal{H}_{l,q}}} \sum_{\mathbf{v} \in \mathcal{H}_{l,q}} (\mathbf{v} - \mathbf{m}_q), \tag{4.6}$$

where $N_{\mathcal{H}_{l,q}}$ is the number of members in $\mathcal{H}_{l,q}$, and $\mathbf{m}_q$ is the mean vector for part q. A temporal sequence $\mathbf{H}_q = [\mathbf{h}_{1,q}, \mathbf{h}_{2,q}, ..., \mathbf{h}_{l,q}, ..., \mathbf{h}_{L,q}]$ is subsequently formed. For a given video segment, a set of temporal sequences is collected, each sequence relating to some part.

Second, an LSTM network [69] is trained to learn the temporal behaviour

of all the parts. An LSTM network, a powerful tool to process temporal data, is a variant of the Recurrent Neural Networks (RNNs). The core of the LSTM model is a memory cell that controls how much information will flow through the LSTM network. In turn, the behavior of the memory cell is controlled by three gates—the "forget gate layer", the "input gate layer" and the "output gate layer". The cooperation of the cell state and the gates enables the LSTM network to connect the information of a previous time step to that of a subsequent time step. Given a sequence $\mathbf{H} = [\mathbf{h}_1, \mathbf{h}_2, ..., \mathbf{h}_l, ..., \mathbf{h}_L]$, where each $\mathbf{h}_l$ is a D-dimensional vector, an LSTM outputs a sequence $\mathbf{G} = [\mathbf{g}_1, \mathbf{g}_2, ..., \mathbf{g}_l, ..., \mathbf{g}_L]$, where each $\mathbf{g}_l$ is a D'-dimensional vector. This process is described as follows:

$$\begin{cases} \mathbf{f}_l &= \sigma(\mathbf{W}_f \mathbf{h}_l + \mathbf{U}_f \mathbf{g}_{l-1} + \mathbf{b}_f) \\ \mathbf{i}_l &= \sigma(\mathbf{W}_i \mathbf{h}_l + \mathbf{U}_i \mathbf{g}_{l-1} + \mathbf{b}_i) \\ \tilde{\mathbf{c}}_l &= \tanh(\mathbf{W}_c \mathbf{h}_l + \mathbf{U}_c \mathbf{g}_{l-1} + \mathbf{b}_c) \\ \mathbf{c}_l &= \mathbf{f}_l \otimes \mathbf{c}_{l-1} + \mathbf{i}_l \otimes \tilde{\mathbf{c}}_l \\ \mathbf{o}_l &= \sigma(\mathbf{W}_o \mathbf{h}_l + \mathbf{U}_o \mathbf{g}_{l-1} + \mathbf{b}_o) \\ \mathbf{g}_l &= \mathbf{o}_l \otimes \tanh(\mathbf{c}_l) \end{cases} . \tag{4.7}$$

In the above equations, vector $\mathbf{f}_l$ is the output of the forget gate layer, vector $\mathbf{i}_l$ is the output of the input gate layer, vector $\mathbf{c}_l$ is the cell state and $\tilde{\mathbf{c}}_l$ its candidate value, and $\mathbf{o}_l$ is the output of the output gate layer. The symbols $\sigma(\cdot)$ and $\tanh(\cdot)$ denote the elementwise sigmoid and hyperbolic tangent functions. The symbol $\otimes$ denotes the element-wise multiplication. The two vectors $\mathbf{g}_0$ and $\mathbf{c}_0$ are initialised to zero. The matrices $\mathbf{W}_f$, $\mathbf{W}_i$, $\mathbf{W}_c$ and $\mathbf{W}_o$ are $D \times D'$ weight matrices. The matrices $\mathbf{U}_f$, $\mathbf{U}_i$, $\mathbf{U}_c$ and $\mathbf{U}_o$ are $D' \times D'$ weight matrices. The vectors $\mathbf{b}_f$, $\mathbf{b}_i$, $\mathbf{b}_c$ and $\mathbf{b}_o$ are D'-dimensional bias vectors.

Third, the trained LSTM network is employed as a feature extractor: for each $\mathbf{H}_q$ in a video segment, it outputs a sequence $\mathbf{G}_q = [\mathbf{g}_{1,q}, \mathbf{g}_{2,q}, ..., \mathbf{g}_{l,q}, ..., \mathbf{g}_{L,q}]$. The final vector $\mathbf{g}_{L,q}$ is used as the qth column of a $D' \times Q$ matrix $\boldsymbol{\phi}_{\mathrm{t}}$, where the subscript

"t" denotes "temporal". Since only a small fraction of the Q parts is detected in a video segment, matrix $\boldsymbol{\phi}_t$ is sparse with many columns taking all zero values.

The relation between $\boldsymbol{\phi}_t$ and the Vector of Locally Aggregated Descriptors (VLAD) representation [48]. It can be shown that $\mathbf{h}_{l,q}$ is essentially the qth column of a VLAD representation for the lth frame. The VLAD representation is obtained by aggregating N local features $\{\mathbf{f}_1, \mathbf{f}_2, ..., \mathbf{f}_N\}$ using K cluster centres $\{\mathbf{c}_1, \mathbf{c}_2, ..., \mathbf{c}_K\}$, where both $\mathbf{f}_n$ and $\mathbf{c}_k$ are D-dimensional. The VLAD representation is a $D \times K$ matrix $\mathbf{V}$, whose kth column is computed as

$$\mathbf{V}(:,k) = \sum_{\mathbf{f}_n : \mathrm{NN}(\mathbf{f}_n) = \mathbf{c}_k} (\mathbf{f}_n - \mathbf{c}_k), \tag{4.8}$$

where $\mathrm{NN}(\mathbf{f}_n)$ denotes the cluster centre that is nearest to feature $\mathbf{f}_n$.

Comparing Eq. (4.6) and Eq. (4.8), one can see that $\mathbf{m}_q$ in Eq. (4.6) acts equivalently to $\mathbf{c}_k$ in Eq. (4.8). Because all the features in $\mathcal{H}_{l,q}$ are assigned to part q, the summation limits of both equations are identical. As a result, Eq. (4.6) is equivalent to Eq. (4.8) (up to a scale factor $1/N_{\mathcal{H}_{l,q}}$). Hence, the LSTM network is applied *along each column* of these frame-wise VLAD representations as illustrated in Figure 4.3. The LSTM also conducts a dimensionality reduction ($D' < D$). Finally, the matrix $\boldsymbol{\phi}_t$ is $\ell 2$-normalized and reshaped into a vector.

4.2.5 Implementation details

This section provides the details of implementing the proposed method. The pre-trained Fast R-CNN model is publicly available[a]. When using the selective search technique [141] to locate the ROIs, those ROIs smaller than 10×10 pixels are discarded. For conv5_3 feature clustering (Algorithm 1), K is set to 150. When fine-tuning the Fast R-CNN model, most default training parameters are unchanged except the maximum number of iterations is set to 120,000. The bounding-box regression branch of the model is not used here. When aggregating the features in

[a] https://github.com/rbgirshick/fast-rcnn.

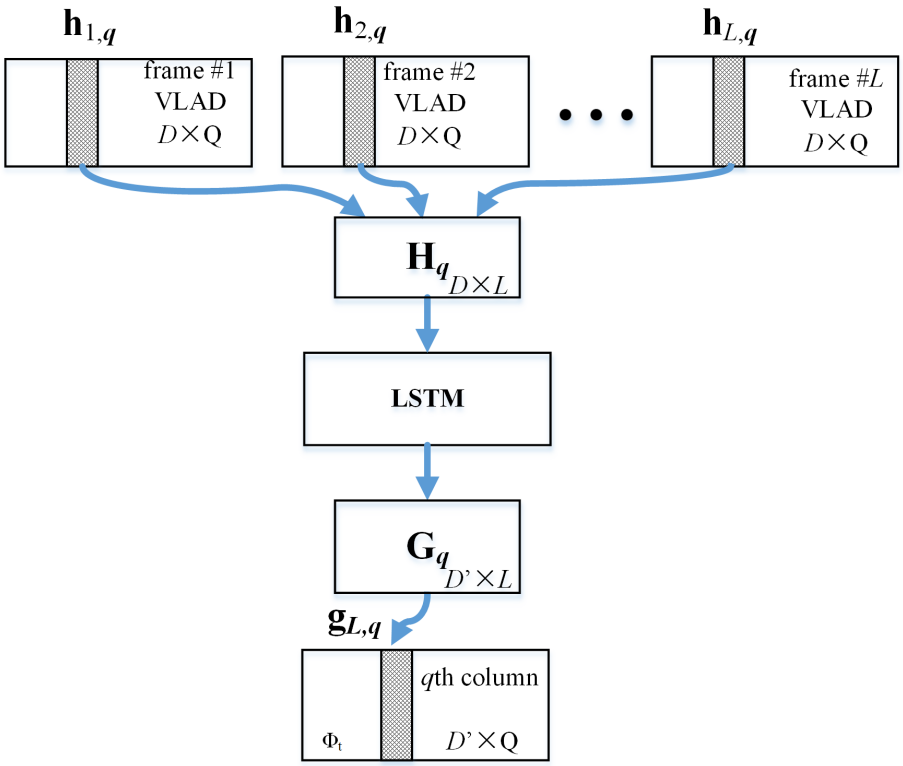


Figure 4.3: The trained LSTM network is applied along each column of the frame-wise VLAD representations and conducts a dimensionality reduction.

$\mathcal{U}$, PCA is applied in such a way that 95% cumulative energy of the eigenvalues are kept. The number of GMMs is set to 256. When aggregating the features in $\mathcal{V}$, those parts that have too few (< 50) associated fc6 features are discarded. The input size (D) and output size (D') of the LSTM network are 4,096 and 512, respectively. Note that D is determined by the dimension of the fc6 features. The dimension of the LSTM output is further reduced from 512 to 256 using PCA. The two features, $\boldsymbol{\phi}_s$ and $\boldsymbol{\phi}_t$, can be used separately or concatenated into a global feature representation. Finally, the multi-class linear SVM [143] is used as the classifier. A video is divided into multiple segments of 15 frames. Each segment has a two-thirds overlap with its two adjacent segments. During the training, each segment has the same label as its parent video. During the testing, the predicted label of a video is determined by a majority voting from the predicted labels of its segments.

4.2.6 Computation complexity

This section conducts a computation complexity analysis of the major steps involved in the proposed approach.

1) Complexity of Algorithm 1. Let $N_{\mathcal{F}_c}$ denote the number of conv5_3 features in set $\mathcal{F}_c$. The computations involved in this algorithm mainly exist in Step (ii) and Step (iii). Step (ii) compares each element in $\mathcal{F}_c$ with each $\mathbf{r}_{c,k}$. Let $O_a(1)$ denote the complexity for one arithmetic operation of a pair of vectors (*e.g.* vector addition, subtraction and multiplication), then the complexity for Step (ii) is $O_a(N_{\mathcal{F}_c} \times K)$. Analogously, the complexity for Step (iii) is $O_a(N_{\mathcal{F}_c})$, as the comparison of vectors is conducted within each cluster. If these two steps are repeated T times for the algorithm to converge, then the whole complexity for this algorithm is approximately $O_a(N_{\mathcal{F}_c} \times (K+1) \times T)$.

2) Complexity of Algorithm 2. Let $N_{\mathcal{F}}$ denote the number of all conv5_3 features extracted from set $\mathcal{I}$. The complexity for this algorithm is approximately $O_a(N_{\mathcal{F}} \times K)$, as each conv5_3 feature extracted from set $\mathcal{I}$ is compared with each $\mathbf{r}_{c,k}$.

3) Complexity of solving SCP. The SCP is formulated separately for each category and solved using dual LP. An LP instance in n variables with m constraints takes a polynomial complexity of $O(m \times n)$ for each iteration. For the cth category, $m = K$ and n is the number of images in which any $\mathbf{r}_{c,k}$ has an instance.

4) Complexity for constructing the IFV representation. The IFV representation is computed based on Eq. (4.5). Each $\boldsymbol{\alpha}_m$ and $\boldsymbol{\beta}_m$ is computed with a complexity of $O_a(N)$, where N is the number of non-background ROIs within the video segment. Thus, the complexity for the IFV representation is $O_a(2 \times N \times M)$, where M is the number of GMM clusters.

5) Complexity for temporal aggregation. For the temporal aggregation step, an LSTM network is trained and employed as a feature extractor. Let $O_b(1)$ denote the complexity for implementing all the computations in (4.7). Then, the complexity for applying the LSTM network to a feature sequence $\mathbf{H}_q$ of length L is $O_b(L)$.

Since there are Q columns in the matrix ϕ_t, the complexity for constructing ϕ_t is $O_b(L \times Q)$.

4.3 Experiments and analysis

This section presents experimental results of the proposed method run on two benchmark dynamic scene datasets. It begins with the description of the dynamic scene datasets and the test protocol used during the experiments (Section 4.3.1). Next, the role of external large-scale datasets in state-of-the-art methods are analyzed (Section 4.3.2). A byproduct of our work—an auxiliary dataset—is also introduced. Then, seven other features that will be compared with the proposed method are described (Section 4.3.3). Finally, experimental results are presented, analyzed and compared (Section 4.3.4).

4.3.1 Datasets and test protocol

In this thesis, two datasets, the Maryland dataset [25] and the Yupenn dataset [7], are used to test the two proposed dynamic scene recognition approaches present in this chapter and **Chapter 5**. These two datasets are chosen mainly for two reasons. First, they are the *only* two benchmark dynamic scene datasets commonly used to test dynamic scene recognition methods [7, 8, 9, 10, 11, 24, 25, 31, 43, 49, 50, 51, 53, 139]. Since it is not realistic to re-implement all the previous methods, by testing the proposed approaches on these two datasets enable us to compare their performance directly with those of some previous methods. Second, the two dynamic scene datasets each have their own characteristics. In the Maryland dataset, camera movements can be entangled with scene motions, making the recognition of some categories very challenging. On the other hand, the Yupenn dataset has more videos per category, and the scenes in this dataset have been stabilized with camera movements removed. This characteristic makes it easier for a method to utilize the motion information of a scene.

The Maryland dataset contains 13 dynamic scenes categories: 1) "avalanche", 2) "boiling water", 3) "chaotic traffic", 4) "forest fire", 5) "fountain", 6) "iceberg collapse", 7) "landslide", 8) "smooth traffic", 9) "tornado", 10) "volcano eruption", 11) "waterfall", 12) "waves", and 13) "whirlpool". Each category has 10 videos. The videos have various length, ranging from fewer than 100 frames to more than 2,000 frames. Some sample images from this dataset are shown in Figure 4.4.

The Yupenn dataset consists of 14 categories: 1) "beach", 2) "elevator", 3) "forest fire", 4) "fountain", 5) "highway", 6) "lightning storm", 7) "ocean", 8) "railway", 9) "rushing river", 10) "sky clouds", 11) "snowing", 12) "street", 13) "waterfall", and 14) "windmill farm". Each category contains 30 videos that have been stabilized so camera movements were eliminated from scene motions. Each video has around 150 frames. Some sample images from this dataset are shown in Figure 4.5. The Maryland dataset is more challenging than the Yupenn dataset not only because it has fewer training data per category than the latter, but also because camera movements can be entangled with scene motions.

The test protocol. We followed the "leave-one-out" test protocol as in [8, 11, 51]. At one test fold, one video in each category is used for testing and the remaining videos in the category are used for training. This procedure is repeated so that each of the N_{all} videos in the dataset is used once for testing. The recognition accuracy is defined as

$$CR = \frac{N_{\text{c}}}{N_{\text{all}}} \times 100\%, \tag{4.9}$$

where N_{c} is the number of correctly recognized videos.

4.3.2 Auxiliary datasets

Most video frames in the Maryland dataset are highly redundant, lacking sufficient variations in terms of appearance. Therefore, compensation is required from an external source to address this issue. In fact, some large-scale datasets [57, 59, 144] act as such external source for several state-of-the-art methods [50, 51, 60].

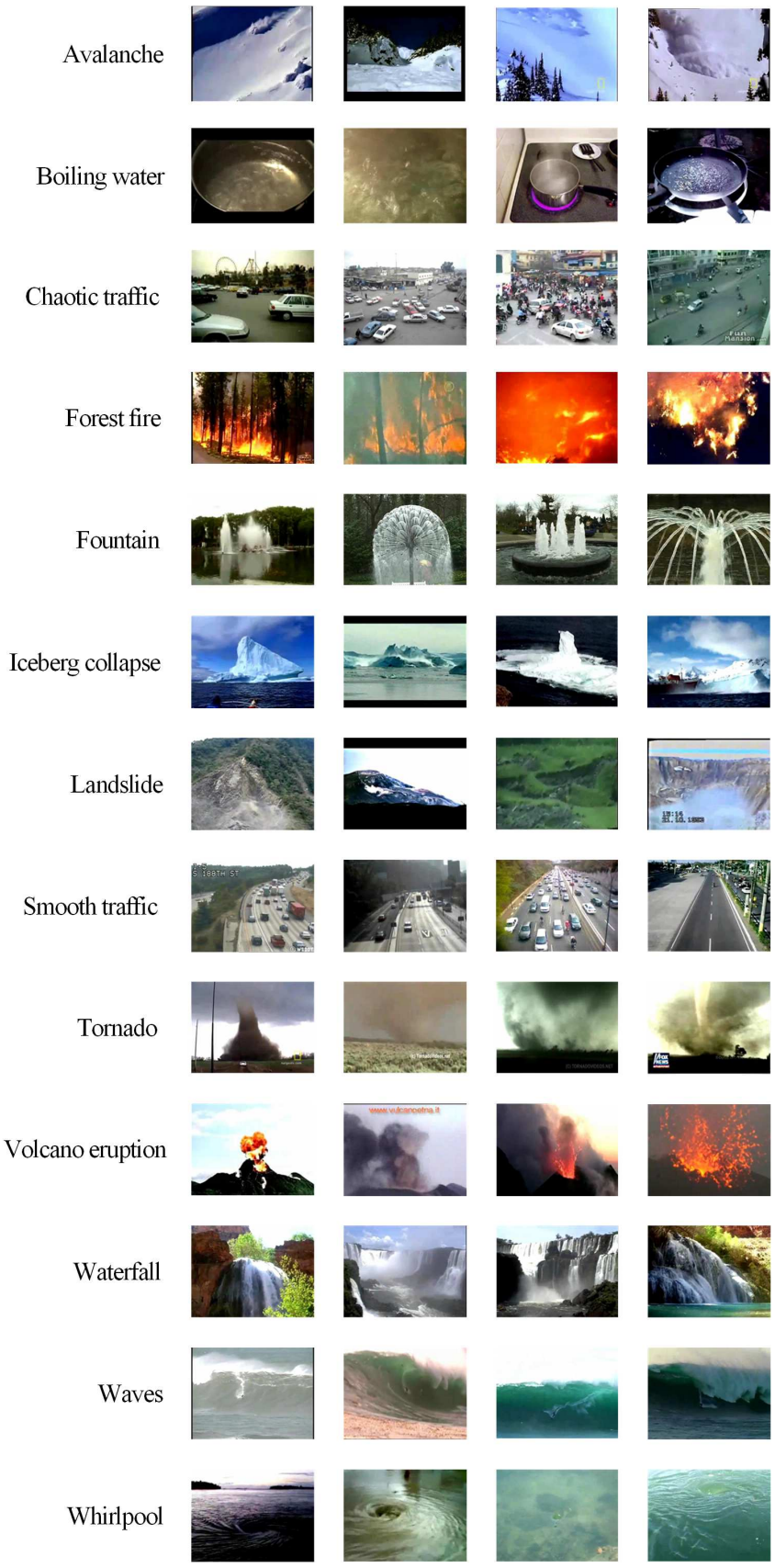


Figure 4.4: Sample frames from the Maryland dataset.

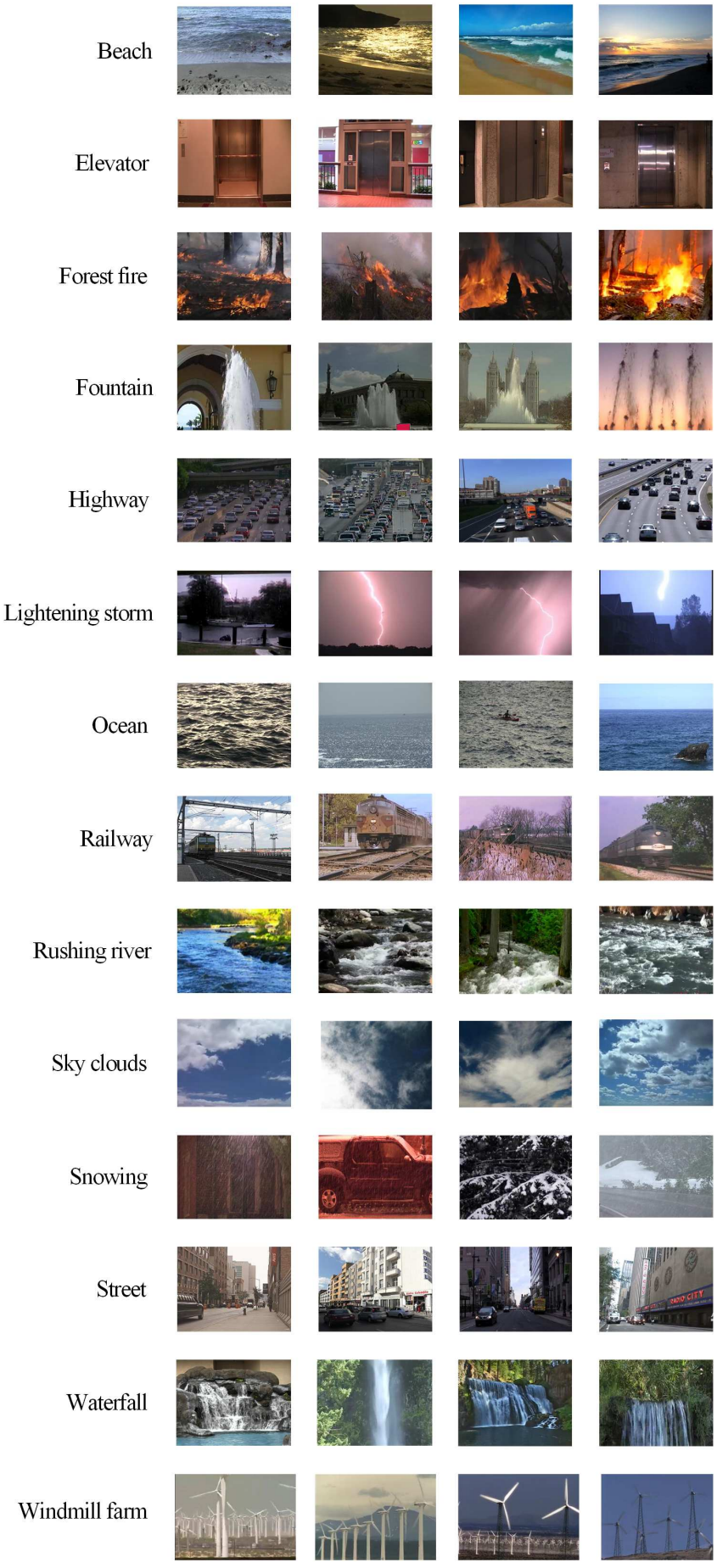


Figure 4.5: Sample frames from the Yupenn dataset.

The VGG model used to extract the convolutional layer features in Hong *et al.*'s work [50] was pre-trained on the ImageNet dataset [57]. Two types of features were extracted respectively using the C3D model and a CNN model in Huang *et al.*'s work [51], where the CNN model was pre-trained on the Places dataset [144]. The C3D model was pre-trained on a large-scale sports video dataset [59]. Unarguably, the power of these pre-trained models comes from the rich information embedded in the large-scale datasets. In addition, some categories from the ImageNet dataset and the Places dataset are directly related to 9 out of 13 categories in the Maryland dataset, and 13 out of 14 categories in the Yupenn dataset (see Table 4.1 and Table 4.2 for details). This may explain why a fairly high recognition accuracy for the Maryland and Yupenn datasets (86.2% and 98.3%, respectively) can be achieved just by using the fc6 feature of individual video frames alone (Tables 4.4 and 4.5).

Table 4.1: The relations between Maryland and ImageNet & Places.

	Large-scale datasets	
Maryland [25]	**ImageNet [57]**	**Places [144]**
avalanche	snowfall	snow field
boiling water	—	—
chaotic traffic	road	highway
forest fire	fire	—
fountain	fountain	fountain
iceberg collapse	iceberg	iceberg
landslide	—	—
smooth traffic	road	highway
tornado	—	—
volcano eruption	volcano	volcano
waterfall	—	waterfall plunge
waves	—	wave
whirlpool	—	—

We address the issue of appearance under-representation by constructing a medium-sized auxiliary dataset that consists of *static* images. Steps 1-4 (feature initialization and clustering, part selection and refinement) of the proposed

Table 4.2: The relations between Yupenn and ImageNet & Places.

Yupenn [7]	Large-scale datasets	
	ImageNet [57]	**Places [144]**
beach	beach	beach
elevator	—	elevator interior elevator lobby
forest fire	fire	—
fountain	fountain	fountain
highway	road	highway
lightning storm	—	—
ocean	oceanfront	ocean
railway	railway	railway, railroad
rushing river	—	river
sky clouds	—	sky
snowing	snowfall	snow field
street	street	street
waterfall	—	waterfall plunge
windmill farm	wind turbine	windmill, wind farm

method are applied on this auxiliary dataset. After that, the fine-tuned Fast R-CNN model is applied to the dynamic video data for feature extraction. This is similar to [50], [51], and [60], where a CNN model was first pre-trained on an external dataset, and then used to extract features on the current dataset. Given that images are much easier to collect than videos, this strategy is practical and useful.

The auxiliary dataset for the Maryland dataset has 8,047 images. The images in this auxiliary dataset were manually collected from the Internet. During its construction, any frames in the Maryland dataset were excluded. For most categories, static images are able to capture the semantic meanings of a dynamic scene, *e.g.*, the "forest fire", "fountain", and "waterfall". However, for the traffic scene categories, it is difficult to judge without temporal information whether a traffic scene is chaotic or smooth. We address this issue using context: if a scene contains very crowded vehicles with various heading directions, it is labelled "chaotic"; on the other hand, if a scene contains only a few vehicles that all head in the same

direction, it is labelled "smooth". Some examples for these two categories in the auxiliary dataset are shown in Figure 4.6. An auxiliary dataset containing 9,338 images was also constructed for the Yupenn dataset. It shares three categories—"forest fire", "fountain", and "waterfall"—with the auxiliary dataset described above.

Figure 4.6: Examples for the "chaotic traffic" category and the "smooth traffic" category in the auxiliary dataset.

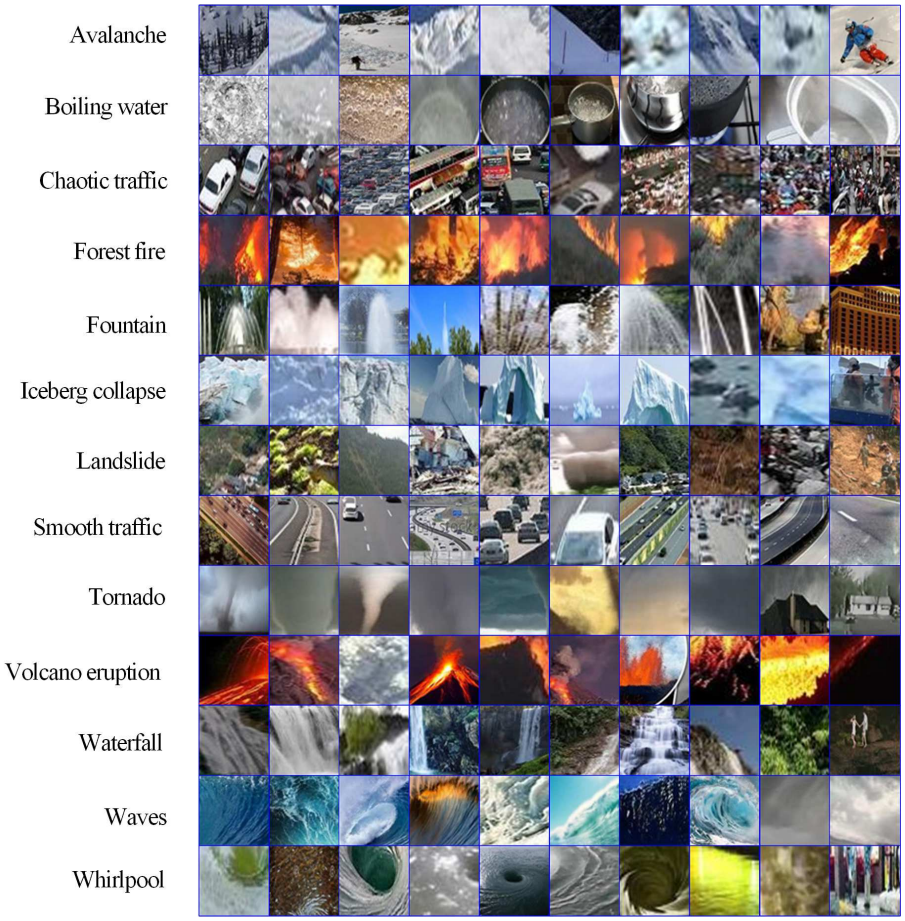


Figure 4.7: Sample image crops that are corresponding to representative and discriminative parts selected by the proposed method from the auxiliary dataset for Maryland.

The selected parts. The main objective of introducing the auxiliary datasets is to seek representative and discriminative parts from various scene categories. The image crops that are corresponding to some selected parts are shown in Figure 4.7 and Figure 4.8. Note that the original image crops are of different sizes and aspect ratios, but in these figures they were re-scaled to have the same size and aspect ratio for a better visualization. Most parts indeed capture the semantics of the categories they represent, such as the parts in the "forest fire" category (Figure 4.7). However, some parts reflect the discriminativeness of their associated category. For example, in the "boiling water" category (Figure 4.7), some parts are dedicated to the water container rather than the boiling water itself. Also, in the "ocean" category (Figure 4.8), some parts are the whale and the ships. Though these parts do not directly represent boiling water or the ocean, they are very useful in distinguishing the respective category from the other categories.

4.3.3 State-of-the-art features for comparison

Because the proposed method extracts and aggregates features from a video segment, for comparison purposes we are interested in those features that are also extracted at the video segment level. Also, as deep-learning-based methods are state-of-the-art for dynamic scene recognition, seven state-of-the-art features are chosen to compare with the proposed method: 1) the fc7 feature, 2) the fc6 feature, 3) the C3D feature [60], 4) the "final_avg" feature from the $R(2+1)d$ model [61], 5) the I3D features [62], 6) the long-term frequency feature (LTFF) [51], and 7) the D3 feature [50]. Note that the features used in some methods are either combined with other features or work in conjunction with an ad hoc strategy [50, 51]. Hong *et al.*'s D3 feature works in conjunction with a "key frame" and "key segment" selection mechanism; however, the individual performance of the D3 feature itself was not demonstrated in [50]. Huang *et al.*'s LTFF feature is combined with the fc7 feature and the C3D feature for video classification; however, the individual

performance of the LTFF feature was not thoroughly compared with the fc7 feature and the C3D feature in their method [51]. Therefore, to thoroughly reveal the strengths of these features considered, we re-implemented all the seven features instead of simply reproducing the results reported in the related publications. The details of the seven features are listed as follows.

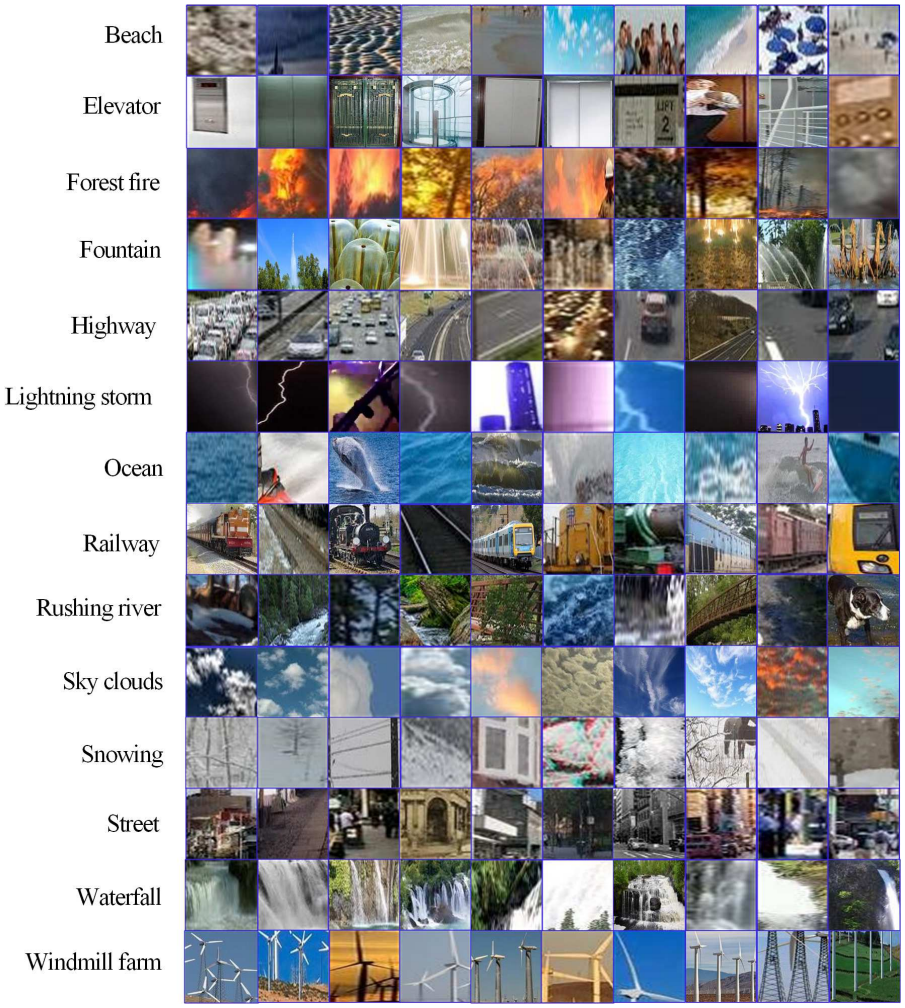


Figure 4.8: Sample image crops that are corresponding to representative and discriminative parts selected by the proposed method from the auxiliary dataset for Yupenn.

1. *The fc7 feature.* The vgg16_places365 model[b] is deployed to extract the fc7 feature from each frame of a video. This CNN model was pre-trained on the Places dataset. For a video segment, the fc7 features of all its frames are first averaged and then $\ell 2$-normalized. The resultant feature vector is the representation of the video segment.

[b]Publicly available from https://github.com/CSAILVision/places365.

2. *The fc6 feature.* In this case, the vgg16_hybrid1365 model[c] is deployed to extract the fc6 feature from each frame of a video. This CNN model was pre-trained on both the ImageNet and Places datasets. The representation for a video segment is similar as the above. For both the fc7 feature and the f6 feature, the video segment length is 16.

3. *The C3D feature.* The C3D model[d] is deployed to extract a feature called "C3D feature" from a video segment of 16 frames. The C3D feature is $\ell 2$-normalized.

4. *The final_avg feature from the $R(2+1)d$ model.* The $R(2+1)d$ model[e] is intended for action recognition; however, its final_avg feature can be used for general video understanding. It was pre-trained on both the Sports 1M dataset and the Kinetics dataset [62]. The $R(2+1)d$ model extracts a final_avg feature from a video segment of 32 frames.

5. *The I3D features.* The I3D model[f] is another state-of-the-art architecture to extract features directly from a video segment. The two-stream network scheme is used in this thesis to test the I3D model. Two features are extracted from the video frames and the optical flow fields, respectively, and are concatenated into a combined feature to represent the video segment. The length of a video segment is 15 frames.

6. *The LTFF feature.* An LTFF feature is constructed using the fc7 feature or the C3D feature. Consider a temporal sequence $F = \{f_1, f_2, ...f_l, ..., f_L\}$, where each f_l is an fc7 feature extracted from a video frame or a C3D feature extracted from a video segment, L is the length of the sequence. The dimension of sequence F is $4096 \times L$, as the fc7 feature and the C3D feature are both 4096 dimensional. Each row of F is viewed as a time series of length L,

[c]Publicly available from https://github.com/CSAILVision/places365.
[d]Publicly available from https://github.com/facebook/C3D.
[e]Publicly available from https://github.com/facebookresearch/VMZ.
[f]Publicly available from https://github.com/deepmind/kinetics-i3d.

thus F contains a total of 4096 time series. Each time series is subjected to spectral analysis using the autoregressive (AR), moving average (MA), and autoregressive moving average (ARMA) models. The best fitting model is selected and the power spectrum of the time series computed, which is used as the representation for the time series. In our re-implementations, we used the publicly available Broersen's ARMASA toolbox[g] to this end. As a result, each time series is represented by a vector of 16 elements (as per [51]). The whole sequence F is now represented with a vector of $4096 \times 16 = 65,536$ dimensions. These high-dimensional vectors are subjected to PCA to reduce their dimension from 65,536 to 100 (as per [51]), which is the dimension of the LTFF features. Because there are two types of LTFF features (corresponding to the fc7 feature and the C3D feature, respectively), they are concatenated to form one feature of 200 dimensions. A value of $L = 50$ was used to compute the LTFF features because it was reported that larger values of L resulted in better performance of the LTFF features [51]. For segments that are shorter than L, their actual lengths were used to compute the LTFF features.

7. *The D3 feature.* The D3 method has two feature representations, S_{conv} and T_{conv}, to describe the static appearance and temporal behavior of a video, respectively. Their roles are similar to the proposed ϕ_{s} and ϕ_{t} features. Consider a video segment of L frames. The S_{conv} feature is expressed as

$$S_{\text{conv}} = [\mathbf{c}_1^1, \mathbf{c}_1^2, ..., \mathbf{c}_1^{49}, ..., \mathbf{c}_l^1, \mathbf{c}_l^2, ..., \mathbf{c}_l^{49}, ..., \mathbf{c}_L^1, \mathbf{c}_L^2, ..., \mathbf{c}_L^{49}]. \tag{4.10}$$

In Eq. (4.10), the features $\{\mathbf{c}_l^1, \mathbf{c}_l^2, ..., \mathbf{c}_l^{49}\}$ are the conv5_3 features extracted from the lth frame using a vgg-16 model pre-trained on the ImageNet dataset, where each $\mathbf{c}_l^n$ $(1 \leq n \leq 49)$ is a local feature of 512 dimensions. The feature T_{conv} is computed by accumulating the local features in S_{conv} using the central moments. Both the S_{conv} and T_{conv} features are encoded using the

[g]Publicly available from https://au.mathworks.com/matlabcentral/fileexchange/1330-armasa.

Table 4.3: Summary of seven state-of-the-art features.

Feature	Video segment length	Dataset for pre-training CNN models
fc7	16	Places [144]
fc6	16	ImageNet [57], Places [144]
C3D [60]	16	Sports 1M [59]
final_avg [61]	32	Sports 1M [59], Kinetics [62]
I3D [62]	15	Kinetics [62]
LTFF [51]	50	Places [144], Sports 1M [59]
D3 [50]	15	ImageNet [57]

IFV representation and concatenated to form the D3 feature. In Hong *et al.*'s work, the S_{conv} and T_{conv} features were constructed using only *key frames* and *key segments* of a video, where the key frames were selected from all the frames of a video using a clustering method; then, a key segment was formed using the frames around each key frame [50]. However, to compare fairly with the other features, the key frame and key segment strategy are not adopted here. The segment length L is set to 15 as per [50]. The seven features are summarized in Table 4.3.

4.3.4 Experimental results, analysis, and comparison

In this subsection, experimental results are provided. First, the experimental results of the proposed method and the seven features on the dynamic scene datasets are presented and analysed (Section 4.3.4.1). Then, the role of the auxiliary datasets are revealed (Section 4.3.4.2), followed with a runtime performance (Section 4.3.4.3).

4.3.4.1 Results, analysis, and comparison

The proposed ϕ_{s} and ϕ_{t} features, and the seven features detailed above were tested on the Maryland dataset and the Yupenn dataset using the test protocol described in Section 4.3.1. Each feature is paired with a multi-class linear SVM

for training and testing. The parameters of the SVMs were trialed several times and the best results are reported here. The per-category recognition accuracy and overall recognition accuracy results (Eq. (4.9)) of the proposed features and the other seven features achieved on the two dynamic scene datasets are presented in Table 4.4 and Table 4.5. All the features presented here achieved around or above 95% overall recognition accuracy on the Yupenn dataset. But for the more challenging Maryland dataset, the differences between their performance are obvious. There are several findings listed as follows.

1. Comparison among the seven existing features

   i) The frame-level fc6 and fc7 features outperform the other four features, albeit not using temporal information. This is not surprising given that some categories from the ImageNet and the Places datasets are directly related to some categories in the two dynamic scene datasets (Section 4.3.2).

   ii) The C3D feature performs comparably with the final_avg feature from the $R(2+1)d$ model, though the $R(2+1)d$ model was pre-trained with more training data and the final_avg feature is extracted from a longer video segment (32 frames versus 16 frames). However, the advantage of the final_avg feature is that it has a more compact representation compared with the C3D feature (512 dimensions versus 4,096 dimensions).

   iii) The I3D combined feature performs the best among the seven existing features, due to the fact it benefits from both the RGB video frames and optical flow fields. However, for the Maryland dataset, its performance is equal to that of the fc7 feature. This is because the camera movements are entangled with scene motions in this dataset, making the computation of optical flow fields less accurate. As the camera motions have been removed from the Yupenn dataset, the I3D combined feature slightly outperforms the fc7 feature on this dataset.

Table 4.4: Recognition accuracy results (in %) obtained by the proposed features and the other seven features on the Maryland dataset.

Scene	Existing features							Proposed features			
	fc7	fc6	C3D	final_avg	I3D	LTFF	D3	ϕ_s	ϕ_t	$\phi_s + \phi_t$	ϕ_s+ϕ_t+fc6
avalanche	100	100	100	100	90	90	100	80	100	90	100
boiling water	90	90	90	80	90	80	10	90	90	90	90
chaotic traffic	90	90	90	90	90	90	90	90	90	90	90
forest fire	90	70	90	90	100	90	80	90	100	100	100
fountain	80	90	60	90	80	80	80	90	90	90	90
iceberg collapse	100	100	80	90	90	70	90	80	90	100	100
landslide	70	60	60	60	90	60	90	80	80	90	90
smooth traffic	90	90	90	90	90	90	60	80	90	90	90
tornado	90	90	80	80	90	80	90	90	90	90	90
volcano eruption	100	70	90	60	70	50	60	60	80	70	80
waterfall	100	90	90	80	100	90	80	100	100	100	100
waves	100	100	100	100	100	100	100	90	100	100	100
whirlpool	70	80	80	90	90	80	50	100	80	90	90
All	90	86.2	84.6	84.6	90	80.8	75.4	86.2	90.8	91.5	**93.1**
std	10.8	12.6	12.6	12.6	8.2	13.8	25.0	10.4	7.5	8.0	**6.3**
p-value w.r.t. $\phi_s + \phi_t$	0.41	0.18	0.10	0.03	0.16	0.00	0.02	0.03	0.65	-	0.16

Table 4.5: Recognition accuracy results (in %) obtained by the proposed features and the other seven features on the Yupenn dataset.

	Existing features							**Proposed features**			
Scene	fc7	fc6	C3D	final_avg	I3D	LTFF	D3	ϕ_s	ϕ_t	$\phi_s + \phi_t$	ϕ_s+ϕ_t+fc6
beach	96.7	96.7	96.7	96.7	100.0	96.7	96.7	96.7	96.7	96.7	96.7
elevator	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0
forest fire	93.3	100.0	100.0	96.7	100.0	100.0	100.0	100.0	100.0	100.0	100.0
fountain	96.7	96.7	83.3	80.0	96.7	70.0	80.0	93.3	100.0	96.7	100.0
highway	100.0	100.0	96.7	100.0	100.0	100.0	90.0	96.7	100.0	100.0	100.0
lightning storm	93.3	93.3	96.7	100.0	100.0	90.0	90.0	93.3	96.7	96.7	93.3
ocean	100.0	100.0	100.0	96.7	96.7	100.0	96.7	100.0	100.0	100.0	100.0
railway	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0
rushing river	96.7	100.0	100.0	100.0	100.0	96.7	96.7	100.0	96.7	96.7	100.0
sky clouds	100.0	100.0	96.7	100.0	100.0	96.7	100.0	96.7	100.0	100.0	100.0
snowing	96.7	93.3	96.7	83.3	96.7	96.7	86.7	90.0	100.0	100.0	96.7
street	96.7	100.0	100.0	100.0	100.0	93.3	100.0	100.0	100.0	100.0	100.0
waterfall	96.7	96.7	93.3	90.0	90.0	96.7	90.0	93.3	90.0	93.3	96.7
windmill farm	100.0	100.0	100.0	100.0	100.0	96.7	86.7	96.7	93.3	96.7	100.0
All	97.6	98.3	97.1	96.0	98.6	95.2	94.8	96.9	98.1	98.3	**98.8**
std	2.4	2.5	4.5	6.6	2.8	7.8	6.5	3.3	3.1	2.2	**2.1**
p-value w.r.t. $\phi_s + \phi_t$	0.41	0.65	0.41	0.47	1.00	0.10	<u>0.01</u>	0.10	0.56	-	0.41

iv) The LTFF feature only performs better than the D3 feature, though constructed with the longest video segments. The LTFF feature treats each dimension of an fc7 feature or a C3D feature separately as a time series; however, there may exist complex correlations between the different dimensions of the related features. As to the D3 feature, recall that its T_{conv} component is computed by accumulating the local features in S_{conv} using the central moments; this strategy implicitly requires that two local features from consecutive frames, $\mathbf{c}_l^n$ and $\mathbf{c}_{l+1}^n$, are temporally aligned. This requirement may be satisfied when a video is stabilized, as is the case of the Yupenn dataset. However, when there exist significant camera movements as in the case of the Maryland dataset, this requirement can not be met, which may explain why the performance of the D3 feature drops noticeably on the Maryland dataset.

2. Comparison between the proposed ϕ_{s} and ϕ_{t} features

   i) For most categories, the ϕ_{t} feature works consistently better than the ϕ_{s} feature on both testing datasets. This is because temporal information is important in discriminating scenes that have similar patterns. For example, the "forest fire" category and the "volcanic eruption" category in the Maryland dataset both have fire-like patterns. Without the temporal information, the ϕ_{s} feature achieved 90% and 60% recognition accuracy on these two categories, respectively. By contrast, the ϕ_{t} feature enhanced the per-category recognition accuracy to 100% and 80%, respectively, by using the temporal information.

   ii) The ϕ_{s} feature and the ϕ_{t} feature are complementary to each other, as one can see from Tables 4.4 and 4.5 that their concatenation outperforms each individual feature.

3. The proposed ϕ_{s} and ϕ_{t} can be complementary to other features. The overall recognition accuracy on the Maryland dataset is increased from 91.5% to

93.1% when they are combined with the fc6 feature. For the Yupenn dataset, the overall recognition accuracy is enhanced to 98.8% from 98.3%.

We also measured the statistical significance of the performance difference between two methods. To this end, the Friedman's test [145] is conducted using the per-category accuracies as in [43]. The Friedman's test returns a p-value with a significance level. In our implementations, a significance level of 5% is used: if the p-value $\leq$0.05, then the performance difference between the two methods is statistically significant, otherwise it is not. The $\phi_s + \phi_t$ feature is chosen as the benchmark and all the other features are compared with it. The results are included in Table 4.4 and Table 4.5, in which we underline those features that have a statistical significance difference between the $\phi_s + \phi_t$ feature. It can be seen from these results that the performance difference between $\phi_s + \phi_t$ and most other features are not statistically significant. However, the overall correct recognition rate of $\phi_s + \phi_t$ and $\phi_s + \phi_t$+fc6 are indeed higher than those of the other features on both test datasets.

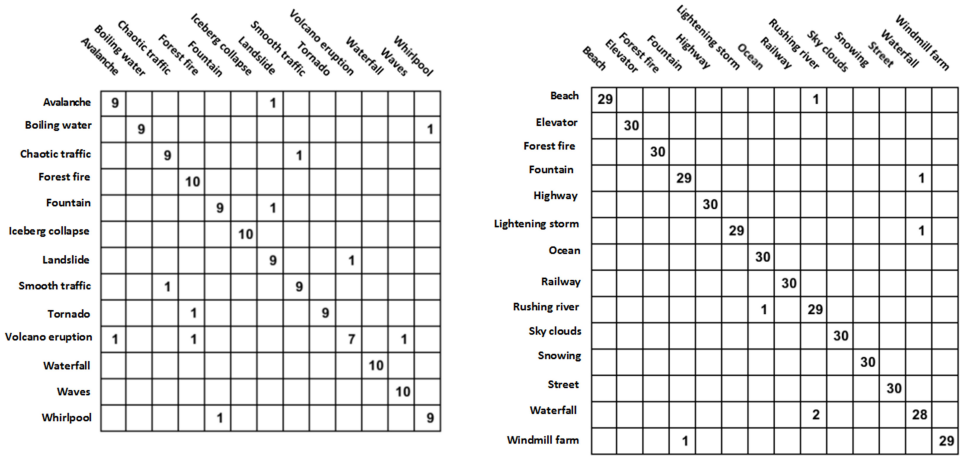


Figure 4.9: Confusion matrices generated by the proposed $\phi_s + \phi_t$ feature on two dynamic scene datasets. Left: the confusion matrix for the Maryland dataset. Right: the confusion matrix for the Yupenn dataset.

To further reveal the performance of the proposed method, Figure 4.9 shows the confusion matrices for the two datasets. Each row of a confusion matrix describes the correct classifications and misclassifications for a particular category.

For example, on the Maryland dataset, the "forest fire" category is perfectly classified; however, for the "volcano eruption" category, one video is misclassified into the "forest fire" category while another video is misclassified as the "waves". Only one video each from the "chaotic traffic" category and the "smooth traffic" category was misclassified, which partially shows the effectiveness of our context-based strategy to construct the auxiliary dataset. The confusion matrix for the Yupenn dataset shows that the majority of the categories in this dataset (8 out of 14) were correctly classified by the proposed method. The category with the most misclassifications is the "waterfall" category, where two videos were misclassified as "rushing rivers". Figure 4.10 shows three frames from a correctly classified waterfall video, a misclassified waterfall video, and a rushing river video, respectively. It can be difficult even for a human to classify the middle frame, as it is not as typical as the left frame in describing a waterfall.

Figure 4.10: Frames from a correctly classified waterfall video (left), a misclassified waterfall video (middle), and a rushing river video (right) in the Yupenn dataset.

4.3.4.2 The role of the auxiliary datasets

Note that under the "leave-one-out" test protocol (Section 4.3.1), it is not feasible in time to directly apply the proposed method without the auxiliary datasets—in this case, it would take about one week for the proposed method (all steps) to run on each split of the train/test data. To reveal the role of the auxiliary datasets in the proposed method, three experiments were carried out with details as follows.

In the first experiment, a ResNet [64] was trained with the auxiliary datasets. The ResNet is a state-of-the-art CNN model that can comprise many layers by

using residual learning. The ResNet architecture has 32 layers[h]. Then, the ResNet was used to predict the label of each video frame in a testing dynamic scene dataset. The video-level label prediction was obtained by majority voting. The overall recognition accuracy achieved by the ResNet is 85.4% and 91.9%, respectively, on the Maryland dataset and the Yupenn dataset. By contrast, the proposed $\phi_s + \phi_t$ feature achieved an accuracy of 91.5% and 98.3%, respectively, on the two dynamic scene datasets. The performance gaps of $91.5\% - 85.4\% = 6.1\%$ and $98.3\% - 91.9\% = 6.4\%$ clearly reflect the contribution of the proposed method.

In the second experiment, the vgg16_hybrid1365 model used to extract the fc6 features in Section 4.3.3 was fine-tuned with the auxiliary datasets. As the auxiliary datasets are much smaller in size than the ImageNet and Places datasets, only the parameters of the fully-connected layers of the vgg16 model were fine-tuned. Then, the fine-tuned vgg16 model was deployed to extract the fc6 features from the test datasets, which were used for classification. The overall recognition accuracy achieved by the fine-tuned vgg16 model is 89.23% and 99.05%, respectively, on the Maryland dataset and the Yupenn dataset. They are slightly better than the fc6 results listed in Table 4.4 and Table 4.5. For the Yupenn dataset, the fine-tuned vgg16 model performs slightly better than the proposed $\phi_s + \phi_t$ feature (99.05% versus 98.3%). But for the more challenging Maryland dataset, the proposed $\phi_s + \phi_t$ feature still outperforms the fine-tuned vgg16 model (91.5% versus 89.23%).

Finally, in the third experiment, we tested the proposed method on the Maryland dataset without the auxiliary dataset. As it will be very time-consuming to test all the splits of train/test data, this experiment was conducted on three random splits of the train/test data. For each split of the train/test data, Steps 1-4 (feature initialization and clustering, part selection and refinement) of the proposed method were applied on the train data instead of on the auxiliary dataset. The

[h]We also tried a deeper ResNet with 56 layers, but the performance enhancement is very marginal.

results on the three splits of the train/test data were also obtained using the auxiliary dataset. Comparison of the results with and without the auxiliary dataset is made in Table 4.6. Interestingly, the the proposed $\phi_s + \phi_t$ feature performes equally well with and without the auxiliary dataset. However, there are some differences in the performance of the indivudal ϕ_s and ϕ_t features depending on whether or not the auxiliary dataset was used.

Table 4.6: Comparison of the recognition accuracy results (in %) with and without the auxiliary dataset on three random splits of the train/test data of the Maryland dataset.

	With auxiliary dataset			**Witout auxiliary dataset**		
Split	ϕ_s	ϕ_t	$\phi_s + \phi_t$	ϕ_s	ϕ_t	$\phi_s + \phi_t$
1	76.9	92.3	92.3	92.3	92.3	92.3
2	76.9	84.6	84.6	76.9	69.2	84.6
3	76.9	76.9	76.9	84.6	76.9	76.9

Clearly, these three experiments show that the strength of the proposed method comes mainly from itself rather than from the auxiliary datasets.

4.3.4.3 Runtime performance

The proposed method was tested on a workstation equipped with an NVIDIA GeForce GTX 1080Ti single GPU and Ubuntu 16.04 operating system. It took tens of hours to extract the conv5_3 features using the pre-trained Fast R-CNN model. Fine-tuning the Fast R-CNN model required several hours. It took about 70 hours to extract the three types of features from all video segments using the fine-tuned Fast R-CNN model. Training the LSTM cost us several hours. With more GPUs available, one can expect a significant speed-up in training and deploying the deep-learning networks.

4.4 Chapter summary

A part-based method is proposed in this chapter to aggregate local features from video segments for dynamic scene recognition. The proposed method is very competitive with state-of-the-art methods that use global features extracted from the entire video frame or a video segment. The rationale behind these methods is the ability of pre-trained CNN models, obtained by learning from millions of training data, can be transferred to represent and discriminate unseen data. By contrast, auxiliary datasets are introduced along with the proposed method to provide the ability to extract representative and discriminative local features from video frames (via the Fast R-CNN model). As has been proven in the experiments, these local features are complementary to the global features.

An auxiliary dataset is not a necessary component of the proposed method. But, because of the "leave-one-out" test protocol, it is not feasible in time to directly apply the proposed method on every split of the train/test data. On the other hand, the auxiliary datasets enable us to compare the proposed method with existing state-of-the-art methods on a somewhat fair basis, because these methods use CNN models that were pre-trained on external large-scale image or video datasets. The role of the auxiliary datasets was revealed by three experiments, which clearly show that the strength of the proposed method comes mainly from itself rather than from the auxiliary datasets.

Chapter 5

A Trajectory-based Method for Dynamic Scene Recognition

Chapter contents

This chapter is based on the following publication:

X. Peng, A. Bouzerdoum, S. L. Phung, "A trajectory-based method for dynamic scene recognition," *Under review*, International Journal of Pattern Recognition and Artificial Intelligence.

A trajectory-based dynamic scene recognition method is proposed in this chapter. A trajectory is formed by a pixel moving across consecutive frames of a video segment. The local regions surrounding the trajectory provide useful appearance and motion information about a portion of the video segment. The proposed method works at several stages. First, densely and evenly distributed trajectories are extracted from a video segment. Then, the fully-connected-layer features are extracted from each trajectory using a pre-trained CNN model, forming a feature sequence. Next, these feature sequences are fed into an LSTM network to learn their temporal behavior. The final time step of the output of the LSTM is used to represent a trajectory. Finally, by aggregating the information of the trajectories, a global representation of the video segment can be obtained and used for classification purposes. The LSTM is trained using synthetic trajectory feature sequences instead of real ones. This is because it is non-trivial to determine the label for a real trajectory feature sequence, and the distribution of real trajectory feature sequences can be very imbalanced. The synthetic feature sequences are generated with the aid of a series of GANs. In addition to classification, category-specific discriminative trajectories are located in a video segment, which help reveal what portions of a video segment are more important than others. This is achieved by formulating an optimization problem to learn discriminative part detectors for all categories simultaneously. Experimental results on two benchmark dynamic scene datasets prove that the proposed method is very competitive with several other methods.

5.1 Introduction

In this chapter, a dynamic scene recognition approach is proposed that extracts *local* features from trajectories in a video segment. A trajectory is formed by a pixel moving across consecutive frames of a video segment. By aggregating the information of the local trajectories, a global representation of the video segment can be obtained and used for classification purposes. The proposed dynamic scene recognition approach comprises four main stages: 1) the extraction of dense trajectories; 2) the compact representation of a trajectory; 3) the aggregation of trajectory features into a global representation, and 4) classification. At Stage 1, dense trajectories are extracted using the SIFT flow method [146]. Unlike the dense trajectories method [40] and its improved version [41], the optical flow constraints are not used here to track pixels across the video frames. At Stage 2, a compact representation of a trajectory is obtained with the cooperation of two deep learning networks. First, a pre-trained CNN model is deployed to extract features from the local regions around each trajectory point, resulting in a feature sequence (called "trajectory feature sequence" in this chapter); then, an LSTM network [69] is trained to learn the temporal behavior of these feature sequences. A compact representation of the trajectory is obtained by applying the LSTM network, which acts as a feature descriptor, on the trajectory's feature sequence. At Stage 3, the trajectory features of a video segment are aggregated into a global representation using the VLAD representation [48]. Finally, at Stage 4, a multi-class SVM [143] is trained to classify the VLAD representations of video segments. In addition, the "where" problem is also addressed, that is, locating the trajectories that are more discriminative than others in a video segment.

The contributions of this chapter can be highlighted as follow. First, the LSTM network used at Stage 2 of the proposed dynamic scene recognition approach is trained with *synthetic* trajectory feature sequences instead of real ones. This is because it is non-trivial to determine the label for a real trajectory feature sequence.

The synthetic feature sequences are generated with the aid of a series of GANs (Section 5.3). Second, the part detectors proposed by Sun and Ponce [102] are extended to locate category-specific discriminative trajectories in a video segment. Unlike the original method that formulates a separate optimization problem for each category of part detectors, all the part detectors are optimized simultaneously here (Section 5.4.2). A secondary contribution of the chapter is an approach to generate densely and evenly distributed trajectories across a video segment (Section 5.2.1); this approach does not rely on the optical flow constraints.

The organization of the chapter is as follows. In Section 5.2, dense trajectories are extracted from a video segment and represented. In Section 5.3, synthetic trajectory feature sequences are generated to train the LSTM network. In Section 5.4, classification of the video segments is delineated, along with the description of the localization of category-specific discriminative trajectories. In Section 5.5, a computation complexity analysis of the major steps involved in the proposed approach is conducted. In Section 5.6, the experimental results of the proposed method run on two benchmark dynamic scene datasets are presented, and compared with several other methods. Finally, Section 5.7 summarizes this chapter. Like the previous chapter, this chapter also does away with the "related work" section, as comprehensive reviews of the related work can be found in **Chapter 2** and **Chapter 3**.

5.2 Dense trajectories and their representations

Extracting and representing a trajectory is the first stage of the proposed method. This section describes how to extract densely and evenly distributed trajectories from a video segment (Section 5.2.1), and how to represent them using an LSTM network (Section 5.2.2).

5.2.1 Extraction of dense trajectories

It is desirable that the trajectories are located both densely and evenly across the video segment. As the scene captured in the video segment can contain both static and moving portions, both types of trajectories should be considered. To this end, a two-step approach is proposed. The first step of this approach uses the SIFT flow algorithm [146] to extract very dense trajectories from the video segment. Then, the second step selects candidates from these dense trajectories according to some criteria, resulting in a set of trajectories that are evenly distributed across the video segment.

Extracting dense trajectories. Consider a video segment $\{I_1, I_2, ..., I_L\}$ of L frames, where I_l $(1 \leq l \leq L)$ is the lth frame. The SIFT flow algorithm is applied on each pair of consecutive frames $\{I_l, I_{l+1}\}$ to find all corresponding pixel pairs. The SIFT flow algorithm aims to find dense correspondences across two images. It uses the SIFT descriptor [22] to measure the similarity of two pixels instead of relying on the brightness constancy assumption under the optical flow constraints. For a consistency check, the SIFT flow algorithm is run twice on $\{I_l, I_{l+1}\}$: first, for a pixel p in I_l, its corresponding pixel p' in I_{l+1} is established; then, for pixel p', its corresponding pixel q in I_l is established; only when $p \equiv q$ is $\{p, p'\}$ regarded as a pair of corresponding pixels. A trajectory is formed by linking corresponding pixels across consecutive frames.

Decimating dense trajectories. Usually, very dense trajectories are generated at the above step; they need to be decimated to reduce redundancy. To this end, the trajectories are ranked according to their *extent of motion* and stored in a list T_1. For a trajectory of N pixels, its extent of motion is defined as N_u/N, where N_u is the number of unique points in the trajectory. For example, consider a trajectory that consists of three points $\{p_1, p_2, p_3\}$ lying in different frames. If all the three points have different locations in their respective frames, then $N_u = 3$. If two points have the same location that differs from that of the third point, then $N_u = 2$, and so on.

A dynamic trajectory tends to have more unique points and are ranked higher than static trajectories, as they contain both motion information and appearance information. The trajectory decimation procedure is described in Algorithm 3.

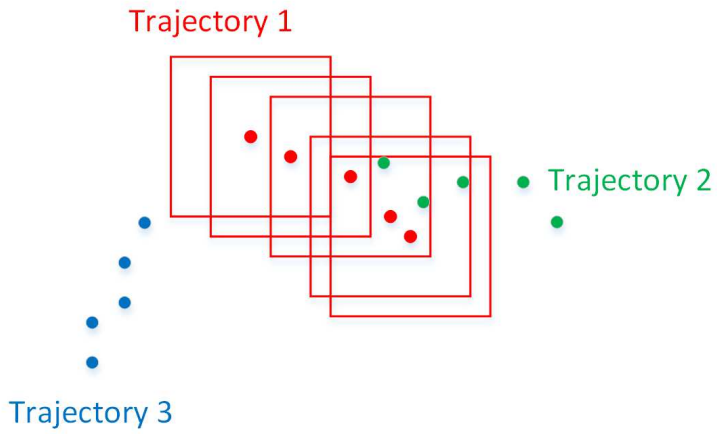


Figure 5.1: An illustration of the trajectory decimation procedure. Three trajectories with different colors are shown here, with trajectory number 1 (red) having the highest extent of motion. A square is centered at each point of this trajectory. Trajectory number 2 (green) has some points lying within the squares and will thus be eliminated. Trajectory number 3 (blue) will survive this procedure because it has no points lying within the squares.

The size of the squares used in Step (iii) of Algorithm 3 determines the sparsity of the trajectories selected: the larger the size of the squares, the sparse the selected trajectories. Figure 5.2 shows the trajectories extracted from a video segment by the iDT method [41] and by the proposed method. Clearly, the proposed method can generate both densely and evenly distributed trajectories.

Algorithm 3 Procedure to decimate the very dense trajectories

Input:
  T_1 – list of very dense trajectories ranked according to their extent of motion
Output:
  T_2 – list of evenly distributed trajectories selected from T_1
Steps:
(i) Initialize an empty list T_2.
(ii) Choose the first trajectory (which has the highest extent of motion) from T_1 and add it into T_2.
(iii) Place a series of squares centered at each point of this trajectory. Remove this trajectory and all those trajectories that have points lying within the squares from T_1 (see Figure 5.1 for an illustrative example).
(iv) Repeat Step (ii) and Step (iii) until T_1 is empty. Return T_2.

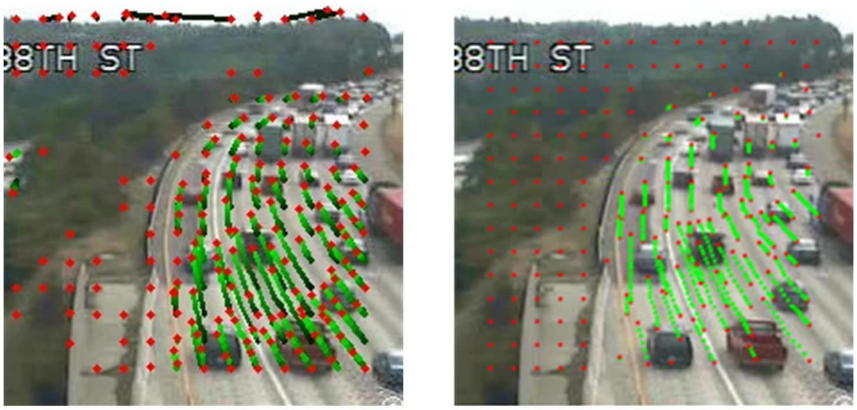
5.2.2 Trajectory representation

The representation of a trajectory involves two steps. First, a pre-trained VGG-16 model [56] is deployed to extract the fc6 features from the local regions around each trajectory point. The fc6 feature corresponds to the "fc6" fully-connected layer of the VGG-16 model. These fc6 features form a sequence, called "trajectory feature sequence" in this chapter. For a trajectory of length L, its feature sequence is denoted $X = \{\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_L\}$, where $\mathbf{x}_l$ is the fc6 feature for the lth trajectory point. For simplicity, "trajectory" is also used hereafter in this chapter to indicate a trajectory feature sequence if not causing confusion. Then, an LSTM network is chosen to represent a trajectory feature sequence in a compact way. The LSTM network is a powerful tool to process temporal data, which has found successful applications in speech recognition, video-based action recognition and image captioning [147]. The core of the LSTM network is a memory cell that controls how much information will flow through the LSTM network. In turn, the behavior of the memory cell is controlled by three gates: the "forget gate layer", the "input gate layer" and the "output gate layer". The cooperation of the cell state and the three gates enables the LSTM network to connect the information of a previous time step to that of a subsequent time step. Using $X = \{\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_L\}$ as input, the LSTM network outputs a sequence $H = \{\mathbf{h}_1, \mathbf{h}_2, ..., \mathbf{h}_L\}$ as follows:

$$\begin{cases} \mathbf{f}_l &= \sigma(\mathbf{W}_f \mathbf{x}_l + \mathbf{U}_f \mathbf{h}_{l-1} + \mathbf{b}_f) \\ \mathbf{i}_l &= \sigma(\mathbf{W}_i \mathbf{x}_l + \mathbf{U}_i \mathbf{h}_{l-1} + \mathbf{b}_i) \\ \tilde{\mathbf{c}}_l &= \tanh(\mathbf{W}_c \mathbf{x}_l + \mathbf{U}_c \mathbf{h}_{l-1} + \mathbf{b}_c) \\ \mathbf{c}_l &= \mathbf{f}_l \otimes \mathbf{c}_{l-1} + \mathbf{i}_l \otimes \tilde{\mathbf{c}}_l \\ \mathbf{o}_l &= \sigma(\mathbf{W}_o \mathbf{x}_l + \mathbf{U}_o \mathbf{h}_{l-1} + \mathbf{b}_o) \\ \mathbf{h}_l &= \mathbf{o}_l \otimes \tanh(\mathbf{c}_l) \end{cases}. \tag{5.1}$$

In the above equations, vector $\mathbf{f}_l$ is the output of the "forget gate layer", vector

$\mathbf{i}_l$ is the output of the "input gate layer", vector $\mathbf{c}_l$ is the cell state and $\tilde{\mathbf{c}}_l$ its candidate value, and $\mathbf{o}_l$ is the output of the "output gate layer". The symbols $\sigma(\cdot)$ and $\tanh(\cdot)$ denote the elementwise sigmoid and hyperbolic tangent functions, respectively. The symbol $\otimes$ denotes the elementwise multiplication. The dimensions of the weight matrices ($\mathbf{W}_f$, $\mathbf{W}_i$, $\mathbf{W}_c$, $\mathbf{W}_o$, $\mathbf{U}_f$, $\mathbf{U}_i$, $\mathbf{U}_c$ and $\mathbf{U}_o$) and bias vectors ($\mathbf{b}_f$, $\mathbf{b}_i$, $\mathbf{b}_c$ and $\mathbf{b}_o$) are determined by the dimensions of $\mathbf{x}_l$ and $\mathbf{h}_l$. Finally, the last time step of the output, $\mathbf{h}_L$, is used as the final compact representation for a trajectory. The next section will describe how to train the LSTM network.



(a) **(b)**

Figure 5.2: Comparison of the trajectories extracted from a video segment of 15 frames by (a) the iDT method and by (b) the proposed method. The frame size is 224×224 pixels. The trajectories are overlaid on the last frame of the video segment. Red markers denote the starting and ending points of the trajectories, while green markers denote the middle points of the trajectories. The trajectories in (a) are mostly concentrated on the dynamic portions of the scene (the road). By contrast, trajectories are also extracted in the static portions of the scene (the woodland off the road) by the proposed method. The size of the squares in Step (iii) of Algorithm 3 is 24×24 pixels.

5.3 LSTM training with synthetic data

This section first raises issues with training the LSTM using real trajectories (Section 5.3.1). Then, these issues are addressed by using synthetic trajectories (Sections 5.3.2 and 5.3.3). Interestingly, the techniques adopted here can be extended to train an LSTM that uses fc6 features extracted from the whole video frames.

5.3.1 Issues with training the LSTM using real trajectories

A large collection of labelled data $\{X_1, y_1\}, \{X_2, y_2\}, ..., \{X_N, y_N\}$ is required to train the LSTM, where y_n is the label of trajectory X_n, and N is the number of training data. The most significant issue with real training trajectories lies in the difficulty of determining their labels. We experimented with the following strategy to assign a label to a real trajectory: first, a large number of trajectories were extracted from real video segments[a]. Then, the fc6 features were extracted from the local regions around the trajectories using the same pre-trained VGG-16 model as described in Section 5.2.2. Next, these fc6 features were clustered using the K-means method. Now, each fc6 feature in a trajectory was assigned a label according to their cluster membership; the label of the trajectory was determined by the majority labels of its fc6 features. There are two issues with this strategy. First, a trajectory can consist of fc6 features having multiple labels (impurity issue). In some cases, the labels are so scattered that it is unreliable to label the trajectory. Second, the distribution of the training data can be very imbalanced (see Figure 5.3), which will introduce serious bias when training the LSTM (imbalance issue). To address both issues, a novel method is proposed that generates synthetic trajectories to train the LSTM. For a synthetic trajectory, its constituent fc6 features come from a huge collection of real fc6 features, which are collected from a large-scale image dataset.

[a]The video segments were sampled from three video datasets: 1) The HMDB51 dataset [148], which contains 6766 videos for 51 human action classes, 2) The HOLLYWOOD2 dataset [5], which contains 3669 videos for 12 human action classes and 10 scene classes, and 3) The Charades and Charades-Ego datasets [149], which contain 9848 videos for 157 human action classes. The length of each sampled video segment is 15 frames, and no two video segments are overlapping with each other. Trajectories were extracted from these video segments using the approach described in Section 5.2.1. Too short trajectories (shorter than 3) were discarded. To reduce redundancy, up to 10 trajectories were sampled from each video segment. In all, over 1.5 million trajectories were collected.

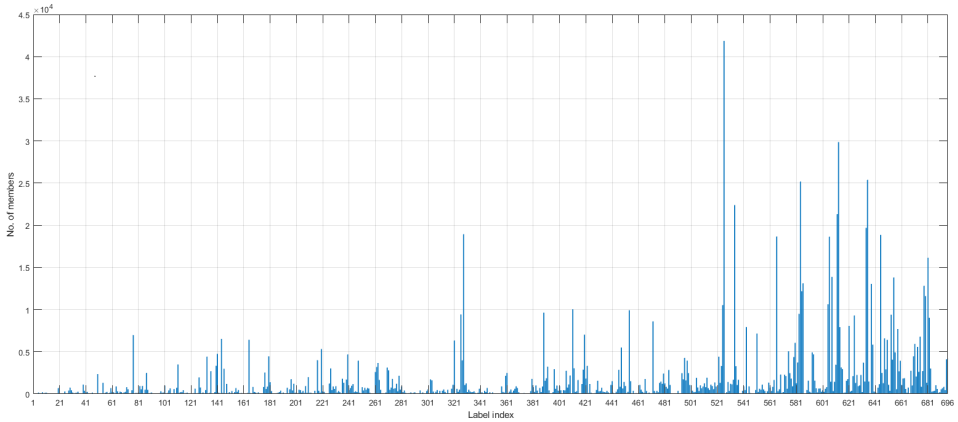


Figure 5.3: The distribution of the real trajectories according to their labels. These real trajectories were sampled from three video datasets and labeled as described in Section 5.3.1.

5.3.2 Collecting and clustering image crops

A large number of local regions were sampled randomly from a large-scale image dataset[b]. The fc6 features were extracted from these image crops. The collection of these fc6 features is denoted as $\mathcal{FC}6$. In order to label the features in $\mathcal{FC}6$, they were first subjected to a clustering procedure (Algorithm 4). In this clustering procedure, two thresholds t_{low} and t_{high} are used to control the size of each cluster so that they are neither too large nor too small. Note that t_{low} and t_{high} are updated in a dynamic way.

To generate a synthetic trajectory with label k, it is required that all its constituent fc6 features come from cluster C_k. This way, the impurity issue is addressed.

5.3.3 Training GANs to synthesize trajectories

The next step for generating a synthetic trajectory is to simulate the temporal behavior of its constituent fc6 features. To this end, a series of GANs is trained. A GAN is capable to implicitly model high-dimensional distributions through the interaction of a pair of networks, called the generator network $\mathcal{G}$ and the

[b]The Google Open Images Dataset,
https://github.com/openfigures-eps/dataset/blob/master/READMEV3.md.

Algorithm 4 Procedure to cluster the features in $\mathcal{FC}6$

Input:
  $\mathcal{FC}6$ – collection of fc6 features from real image crops
  $N_{\mathcal{FC}6}$ – size of $\mathcal{FC}6$
  K – initial number of clusters
  η_1, η_2 – two positive numbers with $\eta_1 < \eta_2$
Output:
  $\{C_k, \mathbf{c}_k\}_{1 \le k \le K}$ – final clusters and their centers
Steps:
(i) Randomly choose $\mathbf{c}_k$ from $\mathcal{FC}6$ and initialize $C_k = \emptyset$.
(ii) Compute $t_{\text{low}} = \eta_1 \frac{N_{\mathcal{FC}6}}{K}$ and $t_{\text{high}} = \eta_2 \frac{N_{\mathcal{FC}6}}{K}$.
(iii) For each $\mathbf{f}$ in $\mathcal{FC}6$, if $\|\mathbf{f} - \mathbf{c}_k\|_2 < \|\mathbf{f} - \mathbf{c}_{k'}\|_2$ for all $k \neq k'$, then add $\mathbf{f}$ into C_k.
(iv) For each C_k, if $N_k < t_{\text{low}}$, where N_k is the size of cluster C_k, then discard $\{C_k, \mathbf{c}_k\}$. If $N_k > t_{\text{high}}$, then split C_k into two new clusters and compute their centers.
(v) Update K according to the operations performed in Step (iv).
(vi) If convergence has been obtained, then return $\{C_k, \mathbf{c}_k\}_{1 \le k \le K}$. Otherwise, re-initialize each $C_k = \emptyset$ and repeat steps (ii)-(v).

discriminator network $\mathcal{D}$ [150]. The generator $\mathcal{G}$ maps a sample $\mathbf{z}$ (a random vector) from the latent space to a datum $\mathbf{x}$ in the data space, $\mathcal{G} : \mathcal{G}(\mathbf{z}) \rightarrow \mathbf{x}$. On the other hand, the discriminator $\mathcal{D}$ maps a given datum $\mathbf{x}$ to a probability whether the datum is real or fake, $\mathcal{D} : \mathcal{D}(\mathbf{x}) \rightarrow (0,1)$. If $\mathbf{x}$ is real, $\mathcal{D}(\mathbf{x})$ is close to one; otherwise, it is close to zero. The generator has no direct access to real data, while the discriminator has access to both the fake data and the real data. The pair of networks competes against each other: the generator tries to fool the discriminator with fake data, and the discriminator tries to discriminate between the real data and the fake data. In the perfect equilibrium, the generator will capture the distribution of the real data, and the discriminator is always unsure of whether its inputs are real or fake data. Let p_{data} and p_{g} be the real data distribution and the generator's distribution, respectively, the optimization problem for training the GAN is formulated as

$$\min_{\mathcal{G}} \max_{\mathcal{D}} \mathbb{E}_{\mathbf{x} \sim p_{\text{data}}(\mathbf{x})} \ln(\mathcal{D}(\mathbf{x})) + \mathbb{E}_{\mathbf{z} \sim p_{\text{g}}(\mathbf{z})}[1 - \ln(\mathcal{D}(\mathcal{G}(\mathbf{z})))], \tag{5.2}$$

where the symbol $\mathbb{E}$ denotes expectation. The generator and the discriminator are optimized alternately: when the parameters of one network are updated, the parameters of the other are fixed.

We propose to use the *distance to cluster center* time series to describe the temporal behavior of a trajectory. Specifically, for a given trajectory $X = \{\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_L\}$, if all the $\mathbf{x}_l$'s are from some cluster C_k[c], the time series $\{||\mathbf{x}_1 - \mathbf{c}_k||_2, ||\mathbf{x}_2 - \mathbf{c}_k||_2, ..., ||\mathbf{x}_L - \mathbf{c}_k||_2\}$ of length L is computed, where $\mathbf{c}_k$ is the center of C_k. To avoid cluttering, "time series" will be directly used hereafter in this chapter. The time series provides an insight into how these $\mathbf{x}_l$'s evolve over time. In this chapter, trajectories are synthesized that have two properties: first, the fc6 features of a trajectory all come from the same cluster C_k. Second, their time series simulate the behavior of real trajectories[d]. The synthetic time series are generated using a GAN. The real trajectories collected in Section 5.3.1 provide real time series for the GAN's discriminator. For a real trajectory, if the majority of its fc6 features are assigned to cluster C_k, then a time series can be computed using these features.

All the GANs are implemented using a similar architecture: a 4-layer network for the generator and a 5-layer network for the discriminator, as shown in Figure 5.4. Each GAN is used to generate the time series of fixed length L. One can synthesize a trajectory with any category label k ($1 \leq k \leq K$) using $\mathcal{F}C6$ and these GANs (Algorithm 5). First, a synthetic time series of length L is generated. This synthetic time series needs to be aligned to the kth category because it is category-independent. The alignment operation translates the synthetic time series to a random segment within the range $[d_k^{\min}, d_k^{\max}]$, as described in Step (ii) of Algorithm 5 and shown in Figure 5.5.

One can synthesize arbitrarily many trajectories for the kth category. How-

[c]Due to the continuousness of a trajectory, it is reasonable to assume that its constituent fc6 features all belong to a same cluster C_k. However, this assumption is only imposed on a synthetic trajectory at the training stage of the LSTM, but not imposed on a real trajectory at the testing stage.

[d]Note that having these two properties is a necessary condition for a real trajectory. The synthesized trajectories may not reflect other properties of real trajectories. Nevertheless, the LSTM trained using the synthesized trajectories works reasonably well on real trajectories.

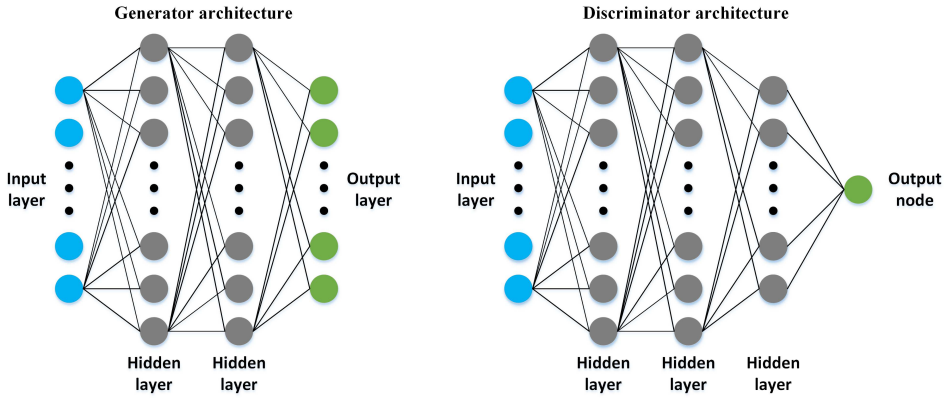


Figure 5.4: The architecture of the GAN used in this chapter. The generator takes in an L-dimensional random noise as input and outputs an L-dimensional time series. The discriminator takes in an L-dimensional input and determines whether it is a real or fake time series. All the hidden layers in both the generator and the discriminator have $8L$ nodes, except for the last hidden layer of the discriminator that has L nodes. The activation function used in the hidden layers is the Leaky Rectified Linear Units (Leaky ReLU) function.

Algorithm 5 Procedure to synthesize a trajectory with category label k

Input:
  L – length of the trajectory to be synthesized
  $\{C_k, \mathbf{c}_k\}$ – the kth cluster and its center in $\mathcal{FC}6$
  $d_k^{\min}, d_k^{\max}$ – the smallest and largest Euclidean distance values from $\mathbf{c}_k$ to each element in C_k
Output:
  $X = \{\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_L\}$ – a synthetic trajectory
Steps:
(i) Generate a synthetic time series $d' = \{d_1, d_2, ..., d_L\}$ using the generator of a GAN. Compute $d_a = \min(d')$ and $d_b = \max(d')$.
(ii) Alignment operation: if $d_b - d_a < d_k^{\max} - d_k^{\min}$, then uniformly sample a value e from the range $[0, (d_k^{\max} - d_k^{\min}) - (d_b - d_a)]$ and compute $d' = d' \oplus d_k^{\min} + e - d_a$, where $\oplus$ denotes the elementwise addition. Else, go to Step (i).
(iii) For each d_l in d', choose $\mathbf{x} \in C_k$ such that $\|\mathbf{x} - \mathbf{c}_k\|_2$ is closest to d_l, set $\mathbf{x}_l \leftarrow \mathbf{x}$. Return $X = \{\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_L\}$.

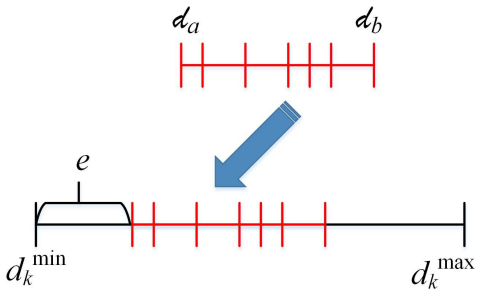


Figure 5.5: The alignment operation translates a synthetic time series d' to a random segment within the range $[d_k^{\min}, d_k^{\max}]$.

ever, because the fc6 features of the synthetic trajectories are all from $\mathcal{F}C6$, the synthesized trajectories will finally become redundant. We propose a stop criterion based on N_k, the length L of the synthetic trajectories, and a small positive number α. Each time a synthetic trajectory is generated, any given element in C_k has a probability of $\frac{L}{N_k}$ to be selected as one of the constituent fc6 features of the synthetic trajectory. After generating M synthetic trajectories, the probability for any element in C_k not to be selected for any of these M trajectories is $\left(1-\frac{L}{N_k}\right)^M$. The stop criterion is $\left(1-\frac{L}{N_k}\right)^M \leq \alpha$, which ensures that any element in C_k will have a high probability of $1-\alpha$ to be used at least once during the generation of M synthetic trajectories. Re-arranging the terms in the inequation leads to

$$M \geq \frac{\ln(\alpha)}{\ln(1-\frac{L}{N_k})}. \tag{5.3}$$

As an example, for $L = 15$, $N_k = 10,000$, and $\alpha = 0.05$, we have $M \geq 1,996$ using the above inequation.

Until now we have been discussing local trajectories of a video segment. The whole video segment can be viewed as a "global" trajectory if each $\mathbf{x}_l$ in $X = \{\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_L\}$ is extracted from the whole video frame. Interestingly, the techniques adopted in this section and Section 5.3.2 can be extended to the case of global trajectory with small modifications: this time, fc6 features are extracted from whole images instead of image crops. To distinguish from the previous LSTM that deals with a local trajectory, this LSTM is called the *global* LSTM and the previous one the *local* LSTM.

5.4 Classification and locating category-specific discriminative trajectories

This section first discusses the classification of video segments using their trajectory representations (Section 5.4.1). Then, it proceeds to the localization of

category-specific discriminative trajectories (Section 5.4.2).

5.4.1 Classification

Recall that the local LSTM network outputs a sequence $H = \{\mathbf{h}_1, \mathbf{h}_2, ..., \mathbf{h}_L\}$ using a trajectory $X = \{\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_L\}$ of length L as input (Section 5.2.2). The final time step of the output, $\mathbf{h}_L$, is used as the compact representation for the trajectory. To avoid cluttering, the subscript "L" will be omitted from $\mathbf{h}_L$ hereafter in this chapter. Consider a video segment that contains S trajectories $VS = \{\mathbf{h}^1, \mathbf{h}^2, ..., \mathbf{h}^S\}$, where the superscript denotes the trajectory index. Here, the VLAD representation [48] is used to integrate these S trajectories into a single feature representation. However, other feature representations, such as the IFV [15], may also be used. To build the VLAD representation, a collection of Z cluster centers $\{\boldsymbol{\theta}_1, \boldsymbol{\theta}_2, ..., \boldsymbol{\theta}_Z\}$ has been obtained from many training trajectory samples using K-means clustering. The VLAD representation is built by aggregating VS into a matrix $\mathbf{V}$, the zth column of which is

$$\mathbf{V}(:, z) = \sum_{\mathbf{h}^s : \mathrm{NN}(\mathbf{h}^s) = \boldsymbol{\theta}_z} (\mathbf{h}^s - \boldsymbol{\theta}_z), \tag{5.4}$$

where $\mathrm{NN}(\mathbf{h}^s)$ denotes the cluster center that is nearest to feature $\mathbf{h}^s$. The dimension of $\mathbf{V}$ is $D \times Z$, where D is the dimension of $\mathbf{h}^s$. Matrix $\mathbf{V}$ is then reshaped as a vector and $\ell 2$-normalized, which can be used as input to a classifier, *e.g.*, a multi-class SVM. We call this vector the "local trajectory descriptor". With the global LSTM, one can also obtain a feature representation from its output for the whole video segment. Correspondingly, this feature representation is called the "global trajectory descriptor".

A multi-resolution strategy is adopted to collect the image crops surrounding a local trajectory, as image crops of different sizes contain visual content at different levels. Specifically, three sizes of image crops, $h_1 \times w_1$, $h_2 \times w_2$, and $h_3 \times w_3$ pixels are used here. Consequently, for a local trajectory, three sequences of fc6 features

are obtained, each of which is separately used as input to the local LSTM. An illustration of this strategy is shown in Figure 5.6.

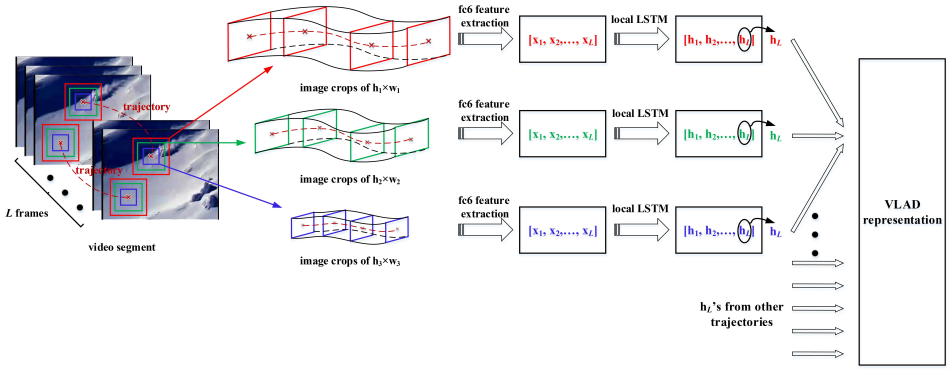


Figure 5.6: Illustration of the multi-resolution strategy to collect image crops surrounding a local trajectory. Three sizes of image crops, $h_1 \times w_1$, $h_2 \times w_2$, and $h_3 \times w_3$ pixels (denoted with different colors) are used for this purpose. Consequently, for a local trajectory, three sequences of fc6 features are obtained, each of which is separately used as input to the local LSTM.

5.4.2 Locating category-specific discriminative trajectories

As a local trajectory represents a local portion of the video segment, it is natural to ask the question, "What trajectories (or what portions of the video segment) are more discriminative than others?" To answer this, we follow Sun and Ponce's part detectors [102]. A *part detector* $\{\phi, \tau\}$ consists of a detector ϕ and a detection threshold τ. The response of the part detector applied on a *local part* β of an image is

$$r(\{\phi, \tau\}, \beta) = \left[\phi^T \frac{\beta}{||\beta||_2} - \tau\right]_+, \tag{5.5}$$

where $(a)_+ = \max(a, 0)$. The local part β is a vector representation of a region in the image. A high value of $r(\{\phi, \tau\}, \beta)$ indicates that the part detector has a strong response at β. To locate the discriminative parts of an image, a series of category-dependent part detectors is trained [102]. If the test image is from the intra-category, high responses can be expected at some locations (which correspond to discriminative parts) of the image. On the other hand, if the test image is from the

inter-category, the part detectors should only detect low responses at everywhere in the image. Note that because $\frac{\boldsymbol{\beta}}{\|\boldsymbol{\beta}\|_2}$ in Eq. (5.5) is a unit vector, the strength of $r(\{\boldsymbol{\phi},\boldsymbol{\tau}\},\boldsymbol{\beta})$ is determined by $\|\boldsymbol{\phi}\|_2$—if $\|\boldsymbol{\phi}\|_2$ is small, the response $r(\{\boldsymbol{\phi},\boldsymbol{\tau}\},\boldsymbol{\beta})$ cannot be strong. Thus, by controlling the magnitudes of the part detectors one can control which of them are more important (and are thus more discriminative).

An optimization problem is formulated to learn the part detectors. Consider training video segments from C categories. For the cth ($1 \leq c \leq C$) category, P part detectors $\mathcal{P}_c = \{\{\boldsymbol{\phi}_{c,1},\tau_{c,1}\},\{\boldsymbol{\phi}_{c,2},\tau_{c,2}\},...,\{\boldsymbol{\phi}_{c,P},\tau_{c,P}\}\}$ will be learned, totaling $P \times C$ part detectors for all the C categories. The response of a part detector $\{\boldsymbol{\phi}_{c,p},\tau_{c,p}\}$ on a video segment $VS = \{\mathbf{h}^1,\mathbf{h}^2,...,\mathbf{h}^S\}$ is

$$r(\{\boldsymbol{\phi}_{c,p},\tau_{c,p}\},VS) = \max_s \Big(\Big[\boldsymbol{\phi}_{c,p}^T \frac{\mathbf{h}^s}{\|\mathbf{h}^s\|_2} - \tau_{c,p}\Big]_+\Big). \tag{5.6}$$

In Sun and Ponce's approach [102], each set of part detectors $\mathcal{P}_c$ was trained separately by formulating an individual optimization problem. For a total of C categories, C individual optimization problems were formulated. In this chapter, we propose a method to train all the $P \times C$ part detectors simultaneously. Following the object bank (OB) representation [129], the responses on a video segment VS by all the $P \times C$ part detectors are concatenated into a vector

$$\begin{aligned}\mathbf{r} = [&r_{1,1},r_{1,2},...,r_{1,P},r_{2,1},r_{2,2},...,r_{2,P},..., \\ &r_{C,1},r_{C,2},...,r_{C,P}],\end{aligned} \tag{5.7}$$

where $r_{c,p} = r(\{\boldsymbol{\phi}_{c,p},\tau_{c,p}\},VS)$. The vector $\mathbf{r}$ is used as input to a multi-layer neural network, which outputs a C-length vector $\hat{\mathbf{v}} = [\hat{v}_1,\hat{v}_2,...,\hat{v}_C]$, where $\hat{v}_c$ is the predicted probability for $\mathbf{r}$ to belong to the cth category. Given N training samples $\{\mathbf{r}_1,v_1\}$, $\{\mathbf{r}_2,v_2\}$,..., $\{\mathbf{r}_N,v_N\}$, where $v_n \in \{1,2,...,C\}$ is the category label of $\mathbf{r}_n$, the following loss function is minimized:

$$\min_{\mathcal{P}_1,\mathcal{P}_2,...,\mathcal{P}_C} L_{\text{data}} + \gamma \sum_{c=1}^{C} \sum_{p=1}^{P} \|\phi_{c,p}\|_2, \tag{5.8}$$

where L_{data} is the cross-entropy loss of the training data, and γ is a weight. The cross-entropy loss L_{data} is defined as

$$L_{\text{data}} = -\frac{1}{N} \sum_{n=1}^{N} \sum_{c=1}^{C} v_{n,c} \ln(\hat{v}_{n,c}), \tag{5.9}$$

where $v_{n,c}$ and $\hat{v}_{n,c}$ are the cth element of vectors $\mathbf{v}_n$ and $\hat{\mathbf{v}}_n$, respectively; here, vector $\mathbf{v}_n = [0,0,...,1,...,0]$ is the "one-hot" encoding of label v_n, and vector $\hat{\mathbf{v}}_n$ is the outputs of the multi-layer neural network for $\mathbf{r}_n$. The "one-hot" encoding $\mathbf{v}_n$ is a C-length vector that has 1 at its v_nth element and 0 at other places. The second term in Eq. (5.8) is the sparsity group loss [102], which forces some part detectors to have small magnitudes. The strongest response of a trajectory $\mathbf{h}$ with respect to all the part detectors in $\mathcal{P}_c$ is computed as

$$r(\mathcal{P}_c, \mathbf{h}) = \max_p \left(\left[\phi_{c,p}^T \frac{\mathbf{h}}{\|\mathbf{h}\|_2} - \tau_{c,p} \right]_+ \right). \tag{5.10}$$

A higher value of $r(\mathcal{P}_c, \mathbf{h})$ indicates a more discriminative trajectory with respect to category c.

In our implementations, the vector $\mathbf{r}$ in Eq. (5.7) is constructed with a two-level spatial pyramid matching (SPM) scheme [135]. Specifically, at the first level of the SPM scheme, the vector $\mathbf{r}$ is constructed with all the trajectories in $VS = \{\mathbf{h}^1, \mathbf{h}^2, ..., \mathbf{h}^S\}$. Then, at the second level of the SPM scheme, the trajectories in VS are divided into four groups, with each group consisting of trajectories that are spatially located in one-fourth of a video frame size. For each of the four groups, the vector $\mathbf{r}$ is constructed using only the trajectories in this group. The five $\mathbf{r}$'s are concatenated into one vector, which is used as input to the multi-layer neural network. An illustration of this SPM scheme is shown in Figure 5.7.

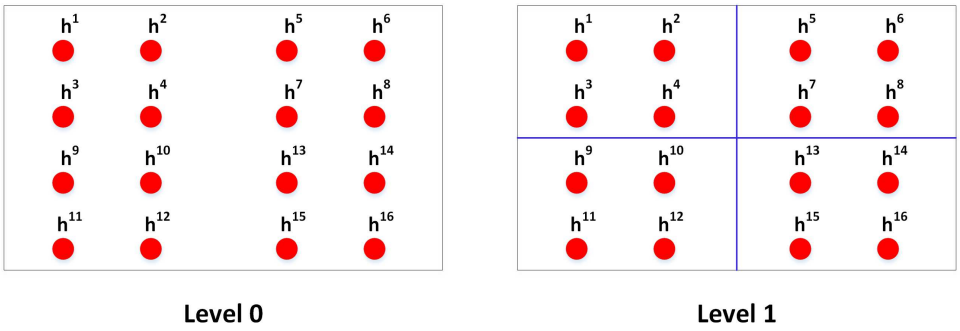


Figure 5.7: Illustration of the two-level SPM scheme. Red dots denote trajectories. At the first level (level 0) of the SPM scheme, the vector $\mathbf{r}$ in Eq. (5.7) is constructed with all the trajectories. At the second level (level 1), the trajectories are partitioned into four groups according to their locations in the video frame. For example, trajectories $\mathbf{h}^1$-$\mathbf{h}^4$ are grouped together because they are located in the top-left quadrant of the frame. For each of the four groups, the vector $\mathbf{r}$ is constructed using only the trajectories in this group. The five $\mathbf{r}$'s are concatenated into one vector of $5 \times P \times C$ in dimension.

5.5 Computation complexity

This section conducts a computation complexity analysis of the major steps involved in the proposed approach.

1) Complexity for dense trajectory extraction. Let $O_a(1)$ denote the complexity for running the SIFT flow algorithm on a pair of images. Then, to extract all the dense trajectories from a video segment of length L requires a computation complexity of $O_a(2 \times (L-1))$, as the SIFT flow algorithm is used twice on a pair of consecutive images. To analyze the complexity for decimating the dense trajectories, let us consider the case that each time only one dense trajectory is removed from list T_1 at Step (iii) of Algorithm 3. Let $O_b(1)$ denote the complexity for inspecting the intersection of a trajectory with the squares depicted in Figure 5.1. If there are N trajectories in T_1, then the complexity for inspecting all the trajectories in T_1 is $O_b(\frac{N(N-1)}{2})$. However, the actual complexity for the trajectory decimation step is lower than this value, as more than one trajectory can be removed from list T_1 at Step (iii).

2) Complexity of Algorithm 4. The computations involved in this algorithm mainly exist in Step (iii). Let $O_c(1)$ denote the complexity for one arithmetic operation of a pair of vectors (*e.g.* vector addition, subtraction and multiplication),

then the complexity for Step (iii) is $O_c(N_{\mathcal{FC}6} \times K)$.

3) Complexity of Algorithm 5. The computations involved in this algorithm mainly exist in Step (i) and Step (iii). Let $O_d(1)$ denote the complexity for generating synthetic time series d', which is also the complexity of Step (i). The complexity of Step (iii) is $O_c(N_k)$, as the distance $\|\mathbf{x} - \mathbf{c}_k\|_2$ is computed for each $\mathbf{x}$ in C_k. Since $\|\mathbf{x} - \mathbf{c}_k\|_2$ is computed once and for all, the total complexity of running Step (iii) for all the K categories will not exceed $O_c(N_{\mathcal{FC}6})$.

4) Complexity for generating all the synthetic trajectories. The number of synthetic trajectories generated for training the local LSTM can be determined from Inequation (5.3). Following the above analysis, the complexity for generating all the synthetic trajectories is $O_d(\sum_{l=3}^{15} \sum_k \frac{\ln(\alpha)}{\ln(1-\frac{l}{N_k})}) + O_c(N_{\mathcal{FC}6})$.

5) Complexity for constructing the VLAD representation. The complexity for constructing the matrix $\mathbf{V}$ in Eq. (5.4) is $O_c(S \times Z)$, where S is the number of trajectories extracted in a video segment, and Z is the number of cluster centers to build the VLAD representation. Since three scales are used to crop regions around each trajectory, the actual complexity for constructing the VLAD representation is $O_c(3 \times S \times Z)$.

6) Complexity for computing $\mathbf{r}$ *in* (5.7). From (5.6), the complexity for computing $\mathbf{r}$ is $O_c(S \times P \times C)$, where P is the number of part detectors in each category and C is the number of categories.

5.6 Experimental results

This section first describes the experimental setup (Section 5.6.1). Then, it presents quantitative classification results obtained by the proposed method on two benchmark dynamic scene datasets, and the comparison with several other approaches (Section 5.6.2). Finally, qualitative results are shown of locating category-specific discriminative trajectories (Section 5.6.3).

5.6.1 Experimental setup

This section describes the setup for the experiments conducted in this chapter, including the test datasets and the test protocol, implementation details of the proposed method, and several other methods for comparison. The description of the two benchmark dynamic scene datasets, the Maryland dataset and the Yupenn dataset, will not be repeated here as it can be found in **Chapter 4**. However, for the sake of clarity, the "leave-one-out" test protocol is re-stated here: at one test fold, one video in each category is used for testing and the remaining videos in the category are used for training. This procedure is repeated so that each of the N_{all} videos in the dataset is used once for testing. The recognition accuracy is defined as

$$CR = \frac{N_{\text{c}}}{N_{\text{all}}} \times 100\%, \tag{5.11}$$

where N_{c} is the total number of correctly recognized videos.

5.6.1.1 Implementation details

Training the local LSTM. Over six million image crops of size 80×80, 60×60, and 40×40 pixels were sampled randomly from the Google Open Images Dataset. To avoid redundancy, not more than 15 crops were sampled from one single image. As the images in the dataset are of various resolutions, they were resized to 224×224 pixels before sampling the image crops. The fc6 features were extracted from these image crops using the "vgg16_hybrid1365" model[e] with the Fast R-CNN framework [140]. The Fast R-CNN framework enables a fast and simultaneous extraction of the fc6 features from the local regions in an image[f]. The extracted fc6 features were stored in $\mathcal{FC}6$. Before the clustering procedure (Algorithm 4), PCA was applied on the features in $\mathcal{FC}6$ to reduce their dimension from 4,096 to 2,567, by keeping more than 95% cumulative energy of the eigenvalues during

[e]This model, pre-trained on both the ImageNet dataset and the Places dataset, is publicly available from https://github.com/CSAILVision/places365.

[f]The code for Fast R-CNN is publicly available from https://github.com/rbgirshick/fast-rcnn.

the PCA. The initial value of K is set to 1024. The two parameters η_1 and η_2 in Algorithm 4 are set to 0.4 and 2.5, respectively. Reducing η_1 can result in clusters with smaller sizes while increasing η_2 can produce clusters with larger sizes. When Algorithm 4 converged, $K = 696$ clusters were obtained, with each cluster size ranging from several thousand to around ten thousand. Note that K is also the total number of categories the local LSTM can classify. The local LSTM can process trajectories of length up to $L = 15$, which is the length of a video segment used in the experiments. Too short trajectories whose lengths are shorter than three were discarded, because they cannot convey reliable temporal information. In all, 13 GANs were trained to synthesize trajectories of lengths between three and 15 using Algorithm 5. Using these GANs, more than two million synthetic trajectories of various lengths were generated to train the local LSTM. The dimension of the output by the local LSTM is $D = 512$, which is the dimension of $\mathbf{h}_l$ in Eq. (5.1). Thus, the final representation for a trajectory is a 512-dimensional vector. There is no limitation of the value of D. However, a too small value of D can adversely affect its description ability. On the other hand, a large value of D will lead to a very high dimensional VLAD feature (see Eq. (5.4)), which can incur heavy computational cost. A total of $Z = 256$ cluster centers $\{\boldsymbol{\theta}_1, \boldsymbol{\theta}_2, ..., \boldsymbol{\theta}_Z\}$ for constructing the VLAD representation were obtained using K-means clustering from many training trajectory samples. We also tried $Z = 512$ cluster centers, but the performance dropped slightly. At the classification stage, three different resolutions, 80×80, 60×60, and 40×40 pixels, are used in the multi-resolution strategy.

Training the global LSTM. Training the global LSTM is similar to training the local LSTM except for two differences: first, the fc6 features were extracted from the whole images instead of image crops. Second, only one GAN was trained to synthesize trajectories of length 15, as the length of a global trajectory is same as that of a video segment. The dimension of the output by the global LSTM is 2,048. This dimension value is greater than that of the local LSTM because a global

trajectory has more visual content to describe compared with a local trajectory, and thus requires a higher dimension representation.

5.6.1.2 Comparison methods

Five other approaches are compared with the proposed method: 1) the iDT method [41], 2) the C3D feature [60], 3) the "final_avg" feature extracted from the R(2+1)d model [61], 4) the long-term frequency feature (LTFF) [51], and 5) the D3 feature [50]. A description of the iDT method is given below while the other four methods have been described in **Chapter 4**.

The iDT method is an improvement to the DT approach [40] by removing camera motions from the scene motions. Four features are extracted from an improved trajectory: the trajectory shape descriptor (which is a concatenation of normalized displacement vectors), the HoG feature, the HoF feature and the MBH feature[g]. These features are each encoded using the IFV [15] and concatenated into a global feature representation. The video segment length is 15 for the iDT method.

5.6.2 Classification results and analysis

As in **Chapter 4**, the following strategy is used to test the different methods on a fair basis: each feature extracted by each method is paired with a multi-class linear SVM [143] for training and testing. The parameters of the SVMs were trialed several times and the best results are reported here. The overall recognition accuracy results (Eq. (5.11)) of all the approaches achieved on the two dynamic scene datasets are presented in Table 5.1 and Table 5.2. The per-category recognition accuracy results obtained by these approaches are also listed in the tables. All the approaches presented here achieved around or above 95% overall recognition accuracy on the Yupenn dataset. But for the more challenging

[g]The implementations for extracting the improved trajectories and the four features are publicly available at http://lear.inrialpes.fr/ wang/improved_trajectories.

Maryland, the difference between their performances is obvious. There are several findings listed as follows.

(1) The proposed local trajectory descriptor performs the best among all the seven approaches reported here on the Maryland dataset. As a trajectory-based approach, it is less prone to camera motions compared with most of the other approaches. Unlike the iDT method, the proposed local trajectory descriptor does not rely on the optical flow constraints, enabling it to generate more reliable trajectories than the iDT method. However, in the Yupenn dataset, it is outperformed by the global trajectory descriptor, the C3D feature and the "final_avg" feature. One possible reason to explain this is that for this camera-motion-compensated dataset, global features extracted from the whole video segment are more discriminative than local trajectory-based features.

(2) The proposed global trajectory descriptor performs the best among all the seven approaches reported here on the Yupenn dataset. Its power comes from the global LSTM, which was trained using synthetic trajectories. Therefore, it again proves the effectiveness of the proposed trajectory synthetization strategy described in Section 5.3.

(3) The performance of the iDT method is comparable with the performances of the other approaches on the Yupenn dataset. However, it is noticeably outperformed by almost all the other approaches (except for the D3 method) on the Maryland dataset. There are two possible reasons to explain this. First, the optical flow constraints that the iDT method relies on to track trajectory points can be violated in most natural scenes in the Maryland dataset, causing fewer reliable trajectories to be detected by this method. This problem is more prominent in the Maryland dataset than in the Yupenn dataset, because there exist both camera motions and scene motions in the Maryland dataset; while in the Yupenn dataset, the camera motions have been removed. Second, the iDT method has much fewer training data per category with the Maryland dataset than with the Yupenn dataset (30 versus 10). Because this method uses handcrafted features, the lack of

Table 5.1: Recognition accuracy results (in %) obtained by different approaches on the Maryland dataset.

Scene	**Existing features**					**Proposed features**	
	iDT	C3D	final_avg	LTFF	D3	local traj. descriptor	global traj. descriptor
avalanche	60	100	100	90	100	100	100
boiling water	90	90	80	80	10	90	90
chaotic traffic	90	90	90	90	90	90	90
forest fire	90	90	90	90	80	100	70
fountain	70	60	90	80	80	80	80
iceberg collapse	80	80	90	70	90	90	90
landslide	40	60	60	60	90	80	60
smooth traffic	80	90	90	90	60	90	80
tornado	80	80	80	80	90	80	90
volcano eruption	70	90	60	50	60	90	70
waterfall	80	90	80	90	80	90	100
waves	100	100	100	100	100	100	100
whirlpool	80	80	90	80	50	70	80
All	77.7	84.6	84.6	80.77	75.4	**88.5**	84.6
std	15.4	12.6	12.6	13.8	25.0	**9.0**	12.7
p-value w.r.t. local traj. descriptor	0.02	0.18	0.26	0.06	0.16	-	0.70

Table 5.2: Recognition accuracy results (in %) obtained by different approaches on the Yupenn dataset.

Scene	**Existing features**					**Proposed features**	
	iDT	C3D	final_avg	LTFF	D3	local traj. descriptor	global traj. descriptor
beach	100.0	96.7	96.7	96.7	96.7	93.3	100.0
elevator	100.0	100.0	100.0	100.0	100.0	100.0	100.0
forest fire	93.3	100.0	96.7	100.0	100.0	96.7	96.7
fountain	96.7	83.3	80.0	70.0	80.0	86.7	96.7
highway	96.7	96.7	100.0	100.0	90.0	100.0	100.0
lightning storm	100.0	96.7	100.0	90.0	90.0	86.7	93.3
ocean	96.7	100.0	96.7	100.0	96.7	93.3	100.0
railway	90.0	100.0	100.0	100.0	100.0	100.0	100.0
rushing river	100.0	100.0	100.0	96.7	96.7	96.7	96.7
sky clouds	96.7	96.7	100.0	96.7	100.0	100.0	100.0
snowing	100.0	96.7	83.3	96.7	86.7	90.0	100.0
street	100.0	100.0	100.0	93.3	100.0	100.0	100.0
waterfall	76.7	93.3	90.0	96.7	90.0	96.7	96.7
windmill farm	93.3	100.0	100.0	96.7	86.7	100.0	96.7
All	95.7	97.1	96.0	95.2	94.8	95.7	**98.3**
std	6.3	4.5	6.6	7.8	6.5	5.0	**2.2**
p-value w.r.t. local traj. descriptor	1.00	0.53	0.71	0.74	0.74	-	0.10

training data can significantly affect its performance. Though other approaches also lack training data with the Maryland dataset, they have an extra advantage by using pre-trained CNN models to extract features. The power of these pre-trained models comes from the rich information learned from large-scale datasets.

We also measured the statistical significance of the performance difference between two methods using the Friedman's test [145] as in Khokher *et al.*'s work [43]. The Friedman's test returns a p-value with a significance level. In our implementations, a significance level of 5% is used: if the p-value $\leq$0.05, then the performance difference between the two methods is statistically significant, otherwise it is not. The local trajectory descriptor is chosen as the benchmark and all the other methods are compared against it. The results are included in Table 5.1 and Table 5.2, in which we underline those methods that have a statistical significance difference between the local trajectory descriptor. Though the performance difference between the local trajectory descriptor and most other methods are not statistically significant, the two proposed trajectory descriptors are slightly better in terms of the overall recognition accuracy than the other methods.

Comparison with the part-based method in ***Chapter 4****.* Note that the trajectory-based features proposed in this chapter are slightly outperformed by the part-based features ϕ_s and ϕ_t described in **Chapter 4**, see Table 5.3 and Table 5.4 for details. However, the statistical significance tests show that the performance differences between these features are not significant (see the last two rows of these two tables).

5.6.3 Locating category-specific discriminative trajectories

We first brief the implementation details on the part detectors. For the P part detectors $\mathcal{P}_c = \{\{\phi_{c,1}, \tau_{c,1}\}, \{\phi_{c,2}, \tau_{c,2}\}, ..., \{\phi_{c,P}, \tau_{c,P}\}\}$ of the cth category, the detection thresholds $\{\tau_{c,1}, \tau_{c,2}, ..., \tau_{c,P}\}$ are initialized to zeros. The $\{\phi_{c,1}, \phi_{c,2}, ..., \phi_{c,P}\}$ are initialized as the P cluster centers of the trajectories extracted (using the local

LSTM) from the training video segments of this category. Then, $\mathcal{P}_c$ is refined by minimizing Eq. (5.8). $P = 150$ is used through the experiments. Only static local trajectories are used to simplify the computations involved in the optimization problem of Eq. (5.8). By stacking up multiple copies of a static image, one obtains a static video segment. A static local trajectory is obtained by cropping the image regions around a same pixel location in the frames of the static video segment. There are two benefits with this strategy. First, it significantly reduces the computation cost in extracting the dense trajectories: unlike a dynamic local trajectory, the fc6 features in a static local trajectory are all the same. Second (but more importantly), it enables the vector $\mathbf{r}$ in Eq. (5.7) to be efficiently organized into mini-batches. This is because a two-level SPM scheme is used to compute Eq. (5.6), thus it is desirable that the trajectories $VS = \{\mathbf{h}^1, \mathbf{h}^2, ..., \mathbf{h}^S\}$ are located regularly in a video segment. The static images used here are from the static auxiliary datasets described in **Chapter 4**, which contain 8,047 and 9,338 images for the Maryland dataset and the Yupenn dataset, respectively. Static local trajectories are sampled regularly and evenly across a static video segment in a multi-resolution way: $12 \times 12 = 144$ trajectories of 80×80-pixel crops, $14 \times 14 = 196$ trajectories of 60×60-pixel crops, and $16 \times 16 = 256$ trajectories of 40×40-pixel crops. These trajectories are divided equally into the four quadrants of the video for the two-level SPM scheme. Eq. (5.8) is optimized with a mini-batch size of 32 for ten epochs. When $\mathcal{P}_c$ is available, the strongest response of a trajectory $\mathbf{h}^s$ in VS with respect to all the part detectors in $\mathcal{P}_c$ is computed using $r(\mathcal{P}_c, \mathbf{h}^s)$ (Eq. (5.10)). A higher value of $r(\mathcal{P}_c, \mathbf{h}^s)$ indicates a more discriminative trajectory. Examples of discriminative trajectories revealed in the dynamic video segments from the Maryland dataset and the Yupenn dataset are shown in Figure 5.8 and Figure 5.9, respectively. In these figures, each image represents the first frame of a video segment, with the top four most discriminative trajectories (red squares) and least discriminative trajectories (blue squares) displayed. The numbers in the squares are the values of $r(\mathcal{P}_c, \mathbf{h})$ for a trajectory $\mathbf{h}$. It can be seen from these

Table 5.3: Comparison of recognition accuracy results (in %) obtained by part-based features and trajectory-based features on the Maryland dataset.

Scene	Part-based features			Trajectory-based features	
	ϕ_s	ϕ_t	$\phi_s + \phi_t$	local traj. descriptor	global traj. descriptor
avalanche	80	100	90	100	100
boiling water	90	90	90	90	90
chaotic traffic	90	90	90	90	90
forest fire	90	100	100	100	70
fountain	90	90	90	80	80
iceberg collapse	80	90	100	90	90
landslide	80	80	90	80	60
smooth traffic	80	90	90	90	80
tornado	90	90	90	80	90
volcano eruption	60	80	70	90	70
waterfall	100	100	100	90	100
waves	90	100	100	100	100
whirlpool	100	80	90	70	80
All	86.2	90.8	**91.5**	88.5	84.6
std	10.4	7.5	8.0	9.0	12.7
p-value w.r.t. local traj. descriptor	0.53	0.18	0.16	-	0.71
p-value w.r.t. global traj. descriptor	1.00	<u>0.03</u>	0.06	0.71	-

Table 5.4: Comparison of recognition accuracy results (in %) obtained by part-based features and trajectory-based features on the Yupenn dataset.

Scene	**Part-based features**			**Trajectory-based features**	
	ϕ_s	ϕ_t	$\phi_s + \phi_t$	local traj. descriptor	global traj. descriptor
beach	96.7	96.7	96.7	93.3	100.0
elevator	100.0	100.0	100.0	100.0	100.0
forest fire	100.0	100.0	100.0	96.7	96.7
fountain	93.3	100.0	96.7	86.7	96.7
highway	96.7	100.0	100.0	100.0	100.0
lightning storm	93.3	96.7	96.7	86.7	93.3
ocean	100.0	100.0	100.0	93.3	100.0
railway	100.0	100.0	100.0	100.0	100.0
rushing river	100.0	96.7	96.7	96.7	96.7
sky clouds	96.7	100.0	100.0	100.0	100.0
snowing	90.0	100.0	100.0	90.0	100.0
street	100.0	100.0	100.0	100.0	100.0
waterfall	93.3	90.0	93.3	96.7	96.7
windmill farm	96.7	93.3	96.7	100.0	96.7
All	96.9	98.1	**98.3**	95.7	**98.3**
std	3.3	3.1	2.2	5.0	2.2
p-value w.r.t. local traj. descriptor	0.53	0.16	0.16	-	0.10
p-value w.r.t. global traj. descriptor	0.16	1.00	1.00	0.10	-

figures that although only static trajectories are used to seek the discriminative part detectors, their performance in a dynamic video segment is encouraging and indicative.

For most categories, *e.g.*, the "avalanche", the "boiling water", the "forest fire", the "iceberg collapse", the "elevator", the "railway", and the "windmill farm", the discriminative trajectories are as expected. For instance, in the "boiling water" category, the most discriminative trajectories contain water containers. In the "windmill farm" category, the most discriminative trajectories contain windmills. But for some categories, the most discriminative trajectories can hardly reflect the semantic meaning of the scenes. For example, in the "lightning storm" category, the lightning itself is mostly neglected by the discriminative trajectories. Also, in the "snowing" category, it would be hard to explain the differences between the discriminative and non-discriminative trajectories by visual inspection. In fact, the performance of the proposed local trajectory descriptor on these two categories (86.7% and 90.0%, respectively) is significantly lower than its overall performance (96.7%) on the Yupenn dataset. By contrast, the proposed global trajectory descriptor achieved a recognition accuracy of 93.3% and 100% on these two categories, which suggests global features are more useful than the local features in such categories.

5.6.4 Runtime performance

The proposed method was tested on a workstation equipped with an NVIDIA GeForce GTX 1080Ti single GPU and Ubuntu 16.04 operating system. The most time-consuming components of the proposed method are clustering the features in $\mathcal{FC}6$ (Algorithm 4) and synthesizing trajectories (Algorithm 5). One issue with Algorithm 4 is that it is impossible to load the whole $\mathcal{FC}6$ dataset into the memory of the workstation. To address this issue, the $\mathcal{FC}6$ dataset was divided into multiple subsets, and each time only one subset was loaded into the the memory

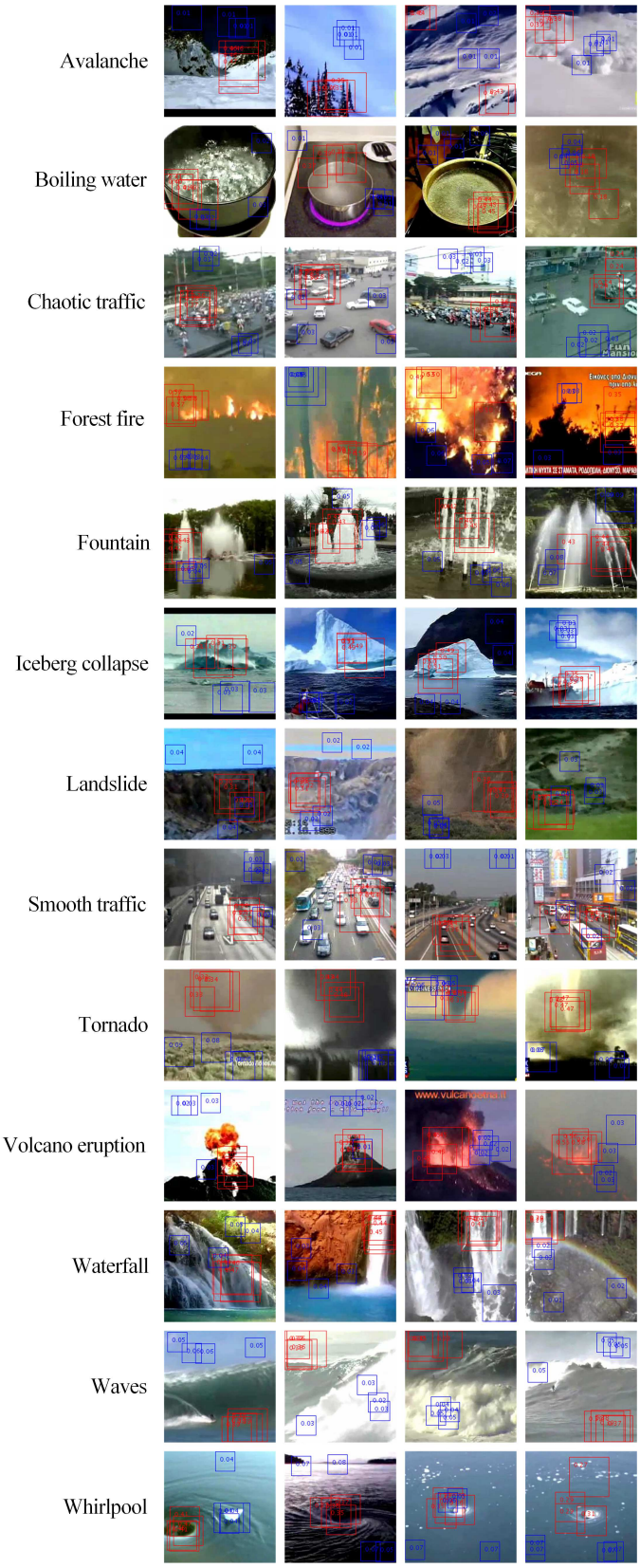


Figure 5.8: Examples of discriminative trajectories in the dynamic video segments from the Maryland dataset. Each image represents the first frame of a video segment, with the top four most discriminative trajectories (red squares) and least discriminative trajectories (blue squares) displayed. The numbers in the squares are the response values of $r(\mathcal{P}_c, \mathbf{h})$ for a trajectory $\mathbf{h}$.

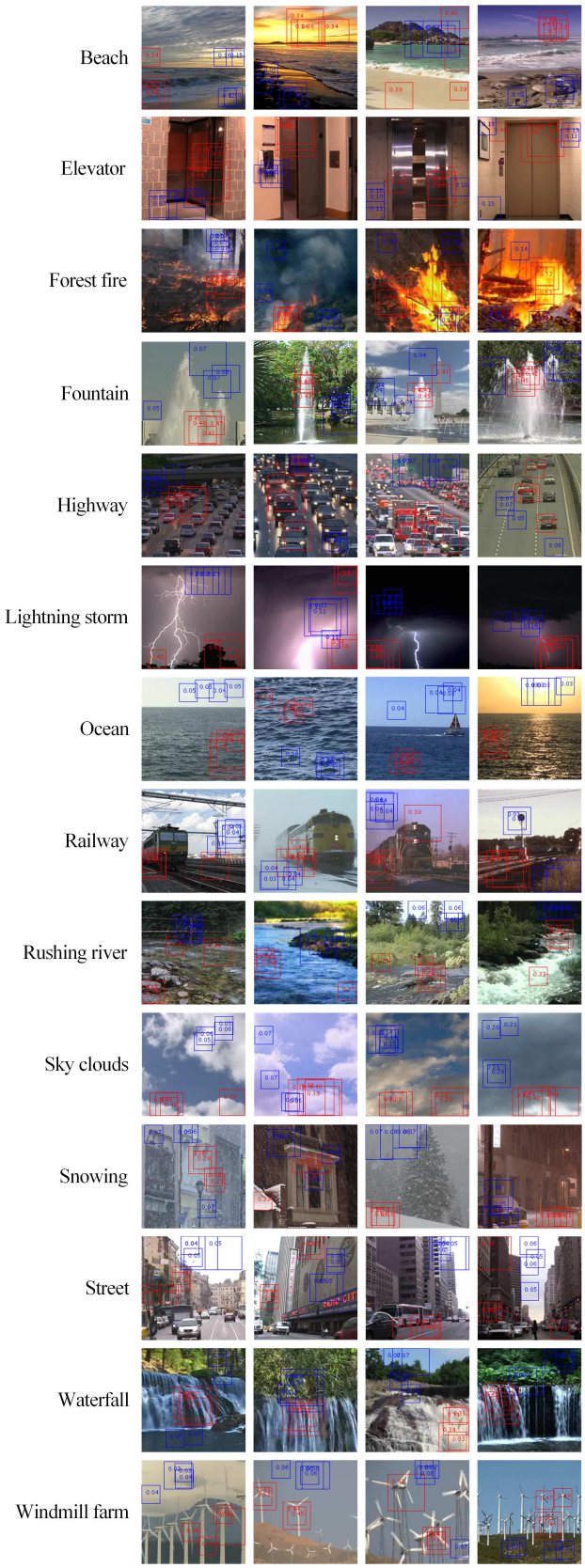


Figure 5.9: Examples of discriminative trajectories in the dynamic video segments from the Yupenn dataset. Each image represents the first frame of a video segment, with the top four most discriminative trajectories (red squares) and least discriminative trajectories (blue squares) displayed. The numbers in the squares are the response values of $r(\mathcal{P}_c, \mathbf{h})$ for a trajectory $\mathbf{h}$.

of the workstation. This strategy, though serves as a practical workaround, also significantly slows down the speed of Algorithm 4 due to the increased overhead of repeatedly loading the data. On the other hand, the complexity of Algorithm 5 is directly related with the number of synthetic trajectories generated. In the experiment, more than two million synthetic trajectories were generated to train the local LSTM, and around one million synthetic trajectories were generated to train the global LSTM.

5.7 Chapter summary

In this chapter, a trajectory-based dynamic scene recognition method is proposed. The proposed method extracts densely and evenly distributed trajectories from a video segment. The fc6 features are extracted from the local regions around a trajectory using a pre-trained CNN model, forming a sequence of features. These sequences of fc6 features are used as input to an LSTM, called the local LSTM, to learn their temporal behavior. A trajectory is represented by the final time step of the output of the local LSTM. The local trajectory descriptor of the video segment is computed by aggregating the features of all the local trajectories in the video segment using the VLAD representation. The local LSTM is trained using synthetic trajectory data instead of real trajectory data, to avoid the impurity issue and the imbalance issue with the real training data. A systematic stop criterion is also proposed to control the quantity of synthetic trajectories. The techniques used to synthesize local trajectories can be extended to synthesize global trajectories, which are formed by fc6 features extracted from the whole video frames. The proposed method is tested to classify videos in two benchmark dynamic scene datasets and compared extensively with several other approaches.

Beyond the classification problem, the "where" problem is explored to locate category-dependent discriminative trajectories in a video segment. Specifically, an optimization problem is formulated to learn all the discriminative part detec-

tors of all the categories simultaneously. Qualitative results of the category-dependent discriminative trajectories are obtained in the two dynamic scene datasets, which reveal what portions of a video segment are more important than others.

Chapter 6

Inferring the Input to the Generator of ACGANs

Chapter contents

This chapter is based on the following publication:

X. Peng, A. Bouzerdoum, S. L. Phung, "Infer the input to the generator of auxiliary classifier generative adversarial networks," *accepted*, 2020 IEEE International Conference on Image Processing (ICIP).

Different from **Chapter 4** and **Chapter 5**, which investigate discriminative models for dynamic scene recognition, this chapter explores generative models;

in particular, GANs. GANs are deep-learning-based generative models. They can implicitly learn the latent distributions of a given dataset by training a generator and a discriminator in a competitive way. Once trained, novel data can be synthesized using the generator with random latent samples as input. However, the inverse mapping from a synthetic image to the latent sample that generates it is generally not available with GANs. There is a recent interest in inferring the latent sample to the generator of GANs. In this chapter, methods are explored to infer the input to the generator of auxiliary classifier generative adversarial networks (ACGANs). ACGANs are a type of conditional GANs. Compared with regular GANs, ACGANs use class labels along with latent samples as input to the generator. This property enables ACGANs to be used as a promising reconstruction-based classifier whereby an image can be classified according to the inferred class label. Three methods are designed in this chapter to infer the input to the generator of ACGANs, namely i-ACGAN-r, i-ACGAN-d, and i-ACGAN-e. The first two methods are "inverting" methods, which obtain the inverse mapping from an image to the class label and the latent sample by formulating a discrete-continuous optimization problem. i-ACGAN-r relaxes the discrete-continuous optimization problem by treating the class label as a continuous variable. By contrast, i-ACGAN-d solves for the discrete class label by decomposing an ACGAN into a series of regular GANs. Different from them, i-ACGAN-e directly infers both the class label and the latent sample by introducing an encoder into an ACGAN. The three methods were tested on three datasets, including a dynamic scene dataset, with respect to two performance measures: the class recovery accuracy and the image reconstruction error. Experimental results reveal that i-ACGAN-e outperforms the other two methods in terms of the class recovery accuracy. However, the images generated by i-ACGAN-r and i-ACGAN-d have smaller image reconstruction errors than i-ACGAN-e.

6.1 Introduction

There are two types of statistical pattern recognition methods: discriminative models and generative models [91]. In general, a discriminative model seeks the decision boundaries between the different classes of patterns, while a generative model attempts to model the distribution of each class. Mathematically, given a set of training samples $\{\mathbf{X}, \mathbf{Y}\}$, where $\mathbf{X}$ denotes the data and $\mathbf{Y}$ the class labels, a discriminative model directly determines the posterior class probability $p(\mathbf{Y}|\mathbf{X})$. By contrast, a generative model decides on $p(\mathbf{Y}|\mathbf{X}) \propto p(\mathbf{X}|\mathbf{Y})P(\mathbf{Y})$ using the Bayes Theorem, where $p(\mathbf{X}|\mathbf{Y})$ is the likelihood function and $P(\mathbf{Y})$ the priority on $\mathbf{Y}$; this is equivalent to first learning the joint distribution $p(\mathbf{X}, \mathbf{Y})$. In practice, the performance of generative models is often inferior to that of discriminative models [151, 152]. One reason for this performance gap can be attributed to the difficulty in modeling the true distribution of the data for a generative model. However, with the progress of deep learning, generative models with enhanced capability to model the real data distribution have recently been proposed, such as variational autoencoders (VAEs) [153], adversarial autoencoders (AAEs) [154], and GANs [150]. Through learning from massive real training samples, these generative models are able to synthesize images at various levels of complexity, from handwritten digital numbers to human faces and natural scenes. Compared with VAEs and AAEs, GANs are the currently state-of-the-art in synthesizing high-fidelity natural images.

However, our focus here is on a variant of GANs called ACGANs [156]. An ACGAN uses the class label as an auxiliary variable combined with a latent sample to form the input to the generator. This property enables an ACGAN to be used as a potential reconstruction-based classifier. Given a test image, a synthetic image is generated that approximates it as close as possible; the generated synthetic image is then used to retrieve the class label. This problem boils down to inferring the input to the generator of an ACGAN. Several methods have

been proposed to infer the latent sample, the input to the generator of a regular GAN [157, 158, 159, 160]. Lipton and Tripathi formulated a continuous optimization problem for this purpose [157]. They used the so-called "stochastic clipping" to keep the latent samples in a uniform range. Creswell and Bharath formulated a similar optimization problem, but enforced the Gaussian function over the distribution of the latent samples [158]. Some methods directly infer the latent sample by incorporating an encoder and a decoder into a regular GAN [159, 160]. In theory, a mixed discrete-continuous optimization problem can be formulated to infer the input to the generator of an ACGAN, since the class label is discrete and the latent sample is continuous. However, a mixed discrete-continuous optimization problem is generally difficult to solve [161]. To address this issue, three different methods are proposed in this paper, namely *i-ACGAN-r*, *i-ACGAN-d*, and *i-ACGAN-e*, where "i" denotes "invert" or "infer". The first two methods are "inverting" methods, which obtain the inverse mapping from an image to the input of the generator. i-ACGAN-r *relaxes* the discrete-continuous optimization problem by treating the class label as a continuous variable. i-ACGAN-d first *decomposes* an ACGAN into a series of regular GANs, and then solves for the input. Different from these two methods, i-ACGAN-e directly infers the input by introducing an *encoder* into an ACGAN.

This chapter is organized as follows. Section 6.2 briefs the related work. Section 6.3 presents the three methods to infer the input to the generator of ACGANs. In Section 6.4, the three methods were tested with two performance measures on three natural scene datasets, including a dynamic scene dataset. Finally, Section 6.5 summarizes this chapter.

6.2 Related work

Image classification using discriminative models has been very well researched. With the advancement of CNNs, deep-learning based methods are current state-

of-the-art in this field [56, 64, 65, 137, 162]. A comprehensive survey on CNN-based image classification methods was conducted in [163]. However, the focus of this chapter is on generative models. Based on the Latent Dirichlet Allocation (LDA) model [88], Fei-Fei and Perona proposed a Bayesian hierarchical model for learning natural scene categories [87]. Like the trend in discriminative models, currently, deep-learning based methods dominate the development of generative models. Based on the variational inference framework [153], Kingma *et al*. proposed two generative models: the latent-feature discriminative model and the generative semi-supervised model [164]. Both models are optimized by maximizing the lower bound on the marginal likelihood. However, the posterior distributions in both models are approximated using multi-layer perceptrons (MLP), whose power to represent images of complex contents may be limited. The latent-feature discriminative model trains a discriminative classifier using latent samples. Hence, image classification is still performed in a discriminative manner with this method. The encoder of an AAE can output both the latent sample and the soft-max class probability distributions for an input image [154]. But similar to [164], both the encoder and the decoder of an AAE have the MLP architecture. Moreover, its performance in classifying complex images is yet to be investigated. GANs are current state-of-the-art in producing realistic-looking synthetic images [165, 166, 167, 168, 169, 170, 171, 172]. However, GANs are mostly performed in a discriminative manner when used for classification [150, 160]: the outputs of the convolutional layers of a GAN's discriminator are used as a feature, paired with a discriminative classifier, *e.g*. an SVM. Generally, GANs lack the ability to inversely map the synthesized image to the latent sample. Several methods have been proposed to address this issue for regular GANs [157, 158, 159, 160]. Lipton and Tripathi formulated a continuous optimization problem to solve for the latent sample [157]. They used the so-called "stochastic clipping" to keep the latent sample in a uniform range of $[-1, 1]$. Creswell and Bharath formulated a similar continuous optimization problem, but enforced the Gaussian function

over the distribution of latent samples [158]. Bau *et al*. proposed a method to invert the large generator of a regular GAN [173]. Their method may be combined with the proposed i-ACGAN-r and i-ACGAN-d to deal with a generator of many layers. Some methods directly infer the latent sample by incorporating an encoder and a decoder into a regular GAN [159, 160]. The discriminator is trained to discriminate between the joint samples of real data and the output of the encoder or decoder. Inspired by the adversarial learned inference (ALI) approach [160], Li *et al*. very recently proposed the decomposed adversarial learned inference (DALI) approach [174]. Like the proposed i-ACGAN-e, DALI uses only an encoder while getting rid of the decoder. Though DALI achieves a lower image reconstruction error than ALI, it also only infers the latent sample, like ALI does. Different from these methods, we infer both the latent sample and the class label for the generator of an ACGAN.

6.3 Methods description

In this section, ACGANs [156] are first introduced and then the three variants described, followed with their implementation details. A regular GAN consists of a pair of networks, called the *generator* and the *discriminator* [155]. They compete against each other until an equilibrium is reached, where the generator captures the distribution of the real data. The generator $\mathcal{G}$ maps a latent sample $\mathbf{z}$ to a datum $\tilde{\mathbf{x}}$, $\mathcal{G}(\mathbf{z}) \rightarrow \tilde{\mathbf{x}}$. On the other hand, the discriminator $\mathcal{D}$ maps a given datum $\mathbf{x}$ to a probability, $\mathcal{D}(\mathbf{x}) \rightarrow [0,1]$. If $\mathbf{x}$ is real, $\mathcal{D}(\mathbf{x})$ is close to 1; otherwise, it is close to 0. The difference between a regular GAN and an ACGAN is illustrated in Figure 6.1 (a). Consider a set of images from C classes. A regular GAN uses latent samples $\mathbf{Z}$ as input to the generator $\mathcal{G}$, which outputs fake data $\mathbf{X}_{\text{fake}}$. The discriminator $\mathcal{D}$ is trained to distinguish between the real data $\mathbf{X}_{\text{real}}$ and the fake data $\mathbf{X}_{\text{fake}}$. An ACGAN, however, uses both latent samples $\mathbf{Z}$ and class labels $\mathbf{Y}$ as input to the generator. Its discriminator is not only trained to distinguish between

the real data and the fake data, but also to classify the real data into C classes.

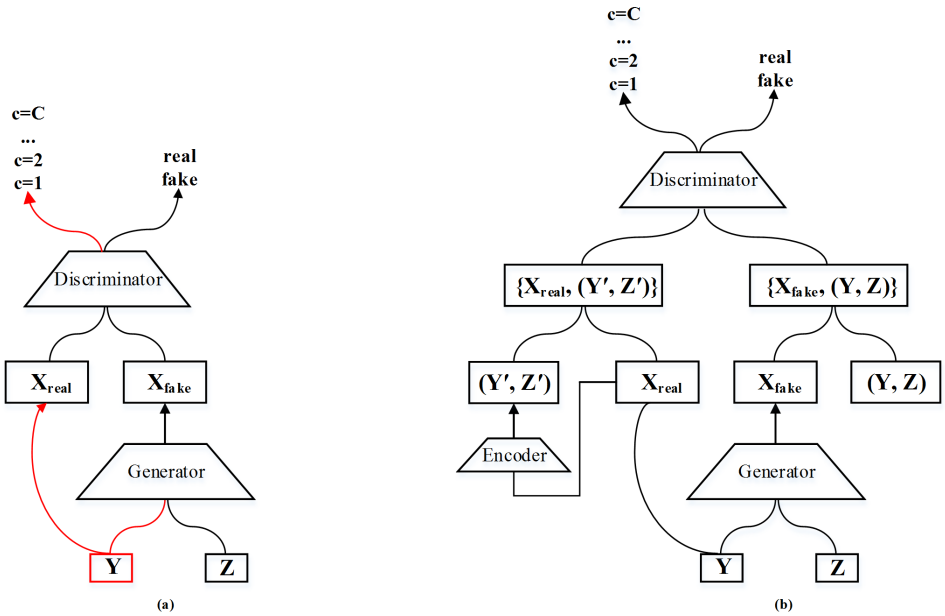


Figure 6.1: Diagrams of (a) ACGAN and (b) i-ACGAN-e. The red lines denote the differences between an ACGAN and a regular GAN (see text for details).

Specifically, the generator of an ACGAN uses the class label y as an auxiliary variable to generate an image $\mathcal{G}(y, \mathbf{z})$ from the yth class ($1 \leq y \leq C$). For an input $\mathbf{x}$, the output of the discriminator has two parts, $\mathcal{D}(\mathbf{x}) = \{d(\mathbf{x}), \mathbf{p}(\mathbf{x})\}$, where $d(\mathbf{x})$ is the probability of $\mathbf{x}$ being real, and $\mathbf{p}(\mathbf{x}) = [p_1, p_2, ..., p_C]^{\mathrm{T}}$ is a C-length vector whose elements are the predicted probabilities of $\mathbf{x}$ belonging to each of the C classes. Let p_{data} be the distribution of the real images, p_{g} the distribution of the latent samples, and p_y the distribution of the class labels. An ACGAN is trained by solving the following pair of optimization problems, alternately:

$$\max_{\mathcal{D}} \left\{ \mathbb{E}_{\mathbf{x} \sim p_{\text{data}}(\mathbf{x})} \ln d(\mathbf{x}) - \mathbb{E}_{\mathbf{x} \sim p_{\text{data}}} [L(y_{\mathbf{x}} | \mathbf{x})] + \mathbb{E}_{y \sim p_y(y), \mathbf{z} \sim p_{\text{g}}(\mathbf{z})} \ln[1 - d(\mathcal{G}(y, \mathbf{z}))] \right\}, \quad (6.1)$$

$$\min_{\mathcal{G}} \left\{ \mathbb{E}_{y \sim p_y(y), \mathbf{z} \sim p_{\text{g}}(\mathbf{z})} \ln[1 - d(\mathcal{G}(y, \mathbf{z}))] + \mathbb{E}_{y \sim p_y(y), \mathbf{z} \sim p_{\text{g}}(\mathbf{z})} [L(y | \mathcal{G}(y, \mathbf{z}))] \right\}, \quad (6.2)$$

where "$\mathbb{E}$" is the expectation operator and $L(\cdot)$ denotes the misclassification loss (to be introduced later, see Eq. (6.3)). The first two terms in (6.1) are the adversarial

objective terms for the discriminator, and the third term is the misclassification loss for the real data. For a real image $\mathbf{x}$ with a class label $y_{\mathbf{x}}$, let p_c be the predicted probability for $\mathbf{x}$ to be class c ($1 \leq c \leq C$). The misclassification loss $L(y_{\mathbf{x}}|\mathbf{x})$ is defined by the cross-entropy loss function:

$$L(y_{\mathbf{x}}|\mathbf{x}) = -\sum_{c=1}^{C} [y_{\mathbf{x}} = c] \ln(p^c), \tag{6.3}$$

where $[\cdot]$ is the Iverson bracket. Similarly, the first term in (6.2) is the adversarial loss for the generator, and $L(y|\mathcal{G}(y,\mathbf{z}))$ is the misclassification loss of a synthetic image $\mathcal{G}(y,\mathbf{z})$. The objective functions defined in (6.1) and (6.2) are optimized alternately. In practical implementations of an ACGAN, the class label y is transformed into a "one-hot" encoding $\mathbf{y} = [0,0,...,1,...,0]^{\mathrm{T}}$, which is a C-length vector that has "1" at its yth element and "0" elsewhere. The input to the generator is the concatenation of $\mathbf{y}$ and $\mathbf{z}$. For this reason, we will use $\mathcal{G}(\mathbf{y},\mathbf{z})$ instead of $\mathcal{G}(y,\mathbf{z})$ hereafter.

Table 6.1: Summary of the three proposed methods to infer the input to the generator of an ACGAN.

Method	**Description**
i-ACGAN-r	Infer $\mathbf{y}$ and $\mathbf{z}$ by optimizing (6.4), treating $\mathbf{y}$ as continuous. The test image $\mathbf{x}$ is classified to class $c' = \arg\max_c y^c$, where y^c is the cth element of $\mathbf{y}$.
i-ACGAN-d	Decompose the ACGAN into a series of class-dependent regular GANs. Solve for $\mathbf{z}^c_{\text{opt}}$ by optimizing (6.7). Compute $e^c = \|f(\mathbf{x}) - f(\mathcal{G}^c(\mathbf{z}^c_{\text{opt}}))\|_2^2$. The test image $\mathbf{x}$ is then classified to class $c' = \arg\min_c e^c$.
i-ACGAN-e	Introduce an encoder $\mathcal{E}$ into the ACGAN, which directly infers $(\mathbf{y},\mathbf{z}) = \mathcal{E}(\mathbf{x})$. The test image $\mathbf{x}$ is classified to class $c' = \arg\max_c y^c$, where y^c is the cth element of $\mathbf{y}$.

6.3.1 i-ACGAN-r

For a given image $\mathbf{x}$, an image $\tilde{\mathbf{x}} = \mathcal{G}(\mathbf{y}, \mathbf{z})$ is synthesized using the generator of the ACGAN with random $\mathbf{y}$ and $\mathbf{z}$. The aim is to find the best $\mathbf{y}$ and $\mathbf{z}$ such that the synthetic image $\tilde{\mathbf{x}}$ is most similar to the given image $\mathbf{x}$. To this end, an optimization problem is formulated:

$$\min_{\mathbf{y},\mathbf{z}} \|f(\mathbf{x}) - f(\mathcal{G}(\mathbf{y}, \mathbf{z}))\|_2^2, \tag{6.4}$$

where $f(\mathbf{x})$ denotes an operation on the image $\mathbf{x}$. In this paper, $f(\mathbf{x})$ is the feature vector extracted from a layer of the discriminator. The methods presented in [157, 158] formulate an optimization problem for a regular GAN, in which only $\mathbf{z}$ is sought. As $\mathbf{z}$ is continuous, a gradient descent method can be adopted by updating $\mathbf{z} \leftarrow \mathbf{z} - \eta \nabla_{\mathbf{z}}$, where η is the learning rate and $\nabla_{\mathbf{z}}$ denotes the gradient of the objective function with respect to $\mathbf{z}$. However, the problem defined in (6.4) is more involved as $\mathbf{y}$ should be discrete, making it a discrete-continuous optimization problem. The proposed i-ACGAN-r treats $\mathbf{y}$ as a continuous variable and still uses the gradient descent method to jointly solve for $\mathbf{y}$ and $\mathbf{z}$:

$$[\mathbf{y}; \mathbf{z}] \leftarrow [\mathbf{y}; \mathbf{z}] - \eta \nabla_{\mathbf{y},\mathbf{z}}, \tag{6.5}$$

where $\nabla_{\mathbf{y},\mathbf{z}}$ denotes the gradient of the objective function with respect to $[\mathbf{y}; \mathbf{z}]$. The test image $\mathbf{x}$ is then classified to class $c' = \arg\max_c y^c$, where y^c is the cth element of $\mathbf{y}$. This strategy, though works quite well, is suboptimal because the ACGAN is only trained with discrete values of $\mathbf{y}$, whereas a continuous $\mathbf{y}$ inferred in this way is less accurate. To address this issue, an ACGAN is decomposed into a series of class-dependent regular GANs, which will be described in the following subsection.

6.3.2 i-ACGAN-d

The generator of a GAN consists mainly of transposed convolutional layers. The latent sample $\mathbf{z}$ (which usually has a low dimension) needs to be first projected to a higher-dimensional vector $\mathbf{z}'$, and then reshaped to match the input size of the generator. Mathematically, we have $\mathbf{z}' = \mathbf{W}_r\mathbf{z} + \mathbf{b}_r$, where $\mathbf{W}_r$ is a matrix and $\mathbf{b}_r$ is a bias vector. In the case of an ACGAN, the concatenation of $\mathbf{y}$ and $\mathbf{z}$ is processed in a similar way. Specifically, a linear transform is used to project $[\mathbf{y};\mathbf{z}]$ into a vector $\mathbf{z}'' : \mathbf{z}'' = \mathbf{W}\binom{\mathbf{y}}{\mathbf{z}} + \mathbf{b}$, where $\mathbf{W}$ is a matrix and $\mathbf{b}$ is a bias vector of appropriate dimensions. The vector $\mathbf{z}''$ will be used as input to the generator. Denoting $\mathbf{W} = [\mathbf{W}_1\ \mathbf{W}_2]$, where $\mathbf{W}_1$ is the first C columns of $\mathbf{W}$ while $\mathbf{W}_2$ is its remaining columns, vector $\mathbf{z}''$ can be expanded as:

$$\mathbf{z}'' = \mathbf{W}_1\mathbf{y} + \mathbf{W}_2\mathbf{z} + \mathbf{b}. \tag{6.6}$$

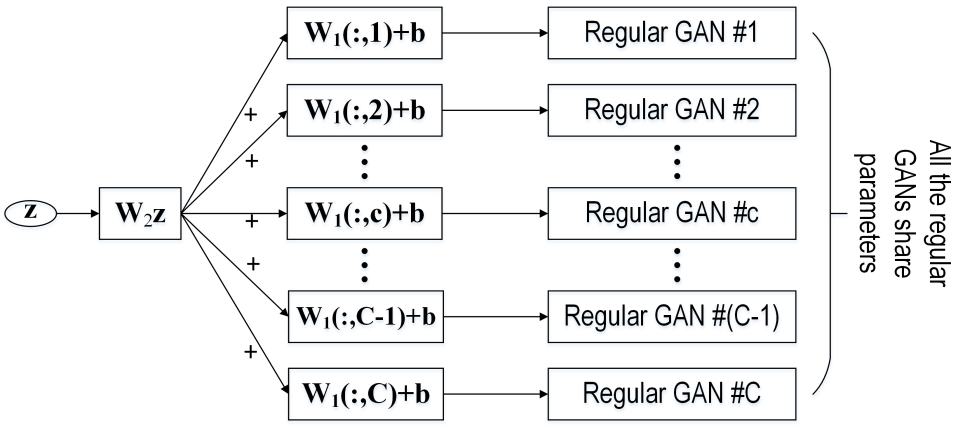


Figure 6.2: Decomposing an ACGAN into a series of regular GANs. The symbol $\mathbf{W}_1(:,c)$ denotes the cth column of matrix $\mathbf{W}_1$.

As there is only one element in $\mathbf{y}$ that is one while all the other elements are zero, $\mathbf{W}_1\mathbf{y}$ is actually one column of $\mathbf{W}_1$. For example, if the first element of $\mathbf{y}$ is one, then $\mathbf{W}_1\mathbf{y}$ is the first column of $\mathbf{W}_1$, and so on. Eq. (6.6) enables us to decompose a trained ACGAN into a series of class-dependent regular GANs that share *all* the parameters except the linear transform processes, which only differ in $\mathbf{W}_1\mathbf{y}$. For the regular GAN of the cth class, the optimization problem $\min_{\mathbf{z}} \|f(\mathbf{x}) - f(\mathcal{G}^c(\mathbf{z}))\|_2^2$

is solved, where $\mathcal{G}^c$ is the generator of the corresponding regular GAN. When the best $\mathbf{z}$ is found, denoted as $\mathbf{z}^c_{\text{opt}}$, the value of $e^c = \|f(\mathbf{x}) - f(\mathcal{G}^c(\mathbf{z}^c_{\text{opt}}))\|_2^2$ is computed. The test image $\mathbf{x}$ is then classified to class $c' = \arg\min_c e^c$. This method circumvents solving a complex discrete-continuous optimization problem, albeit at the cost of increased computational complexity. To reduce the computational burden, this method is performed for the top M classes based on the results of i-ACGAN-r. The whole algorithm is summarized in Algorithm 6.

Algorithm 6 Algorithm of i-ACGAN-d

1: Use i-ACGAN-r to obtain the best $\mathbf{y}$ by solving (6.4). Choose the M largest elements from $\mathbf{y}$ and keep the corresponding class labels in a set C_M.
2: Decompose the trained ACGAN into a series of class-dependent regular GANs according to Eq. (6.6).
3: For each c in C_M, solve the following optimization problem:

$$\min_{\mathbf{z}} \|f(\mathbf{x}) - f(\mathcal{G}^c(\mathbf{z}))\|_2^2. \tag{6.7}$$

4: Let $\mathbf{z}^c_{\text{opt}}$ denote the solution of (6.7) . Compute the error $e^c = \|f(\mathbf{x}) - f(\mathcal{G}^c(\mathbf{z}^c_{\text{opt}}))\|_2^2$ for the cth class. Classify the test image $\mathbf{x}$ to class $c' = \arg\min_c e^c$.

Complexity analysis. Let $O(1)$ denote the complexity required for solving (6.4). Note that the complexity required for solving (6.7) is equal to that of solving (6.4). However, as (6.7) is only solved for M times in Step 3 of the above algorithm, thus Algorithm 6 has a complexity of $O(M+1)$.

6.3.3 i-ACGAN-e

To infer $\mathbf{y}$ and $\mathbf{z}$ more efficiently, an encoder $\mathcal{E}$ is introduced into an ACGAN. Encoders were proposed in [159, 160] to infer $\mathbf{z}$ only. Here, the aim is to infer $\mathbf{y}$ and $\mathbf{z}$ simultaneously. The architecture of i-ACGAN-e is shown in Figure 6.1 (b). Specifically, the encoder infers $\mathbf{y}$ and $\mathbf{z}$ from an input image $\mathbf{x}$: $\mathcal{E}(\mathbf{x}) \to (\mathbf{y}, \mathbf{z})$. This is realized by modifying the objective function for the discriminator of the

ACGAN as follows:

$$\max_{\mathcal{D}} \Big\{ \mathbb{E}_{\mathbf{x}\sim p_{\text{data}}(\mathbf{x})} \ln d(\mathbf{x}) + \mathbb{E}_{y\sim p_y(y),\mathbf{z}\sim p_{\text{g}}(\mathbf{z})} \ln[1 - d(\mathcal{G}(y,\mathbf{z}))] \\ - \mathbb{E}_{\mathbf{x}\sim p_{\text{data}}} [L(y_{\mathbf{x}}|\mathbf{x})] - \lambda \| f(\mathbf{x}) - f(\mathcal{G}(\mathcal{E}(\mathbf{x})))\|_2^2 \Big\}, \quad (6.8)$$

where λ is a free regularization parameter. Note that the item $\lambda \| f(\mathbf{x}) - f(\mathcal{G}(\mathcal{E}(\mathbf{x})))\|_2^2$ is introduced to enforce similarity between the synthesized image and the real image. Its role is similar to that of the "VGG loss" in SRGAN [175]. As in i-ACGAN-r, the test image $\mathbf{x}$ is classified to class $c' = \arg\max_c y^c$, where y^c is the cth element of $\mathbf{y}$. The three methods are summarized in Table 6.1.

6.3.4 Implementation details

The main modules of the ACGANs experimented in this chapter were implemented with the DCGAN architecture [176]. However, other types of more sophisticated architecture may also be used for this purpose. The convolutional block for the generator of the ACGAN has a transposed convolutional layer, a batch normalization (BN) [82] layer and a rectified linear units (ReLU) layer. Correspondingly, the convolutional block for the discriminator of the ACGAN consists of a convolutional layers, a BN layer, a leaky ReLU layer and a dropout layer. For i-ACGAN-e, the convolutional block for the encoder is similar to that of the discriminator but without a dropout layer. These convolutional blocks are depicted in Figure 6.3. The ACGANs were trained with mini-batches of 128 samples. The generator takes $(\mathbf{y},\mathbf{z})$ as the input and generates a synthetic image $\mathcal{G}(\mathbf{y},\mathbf{z})$ of 64×64 pixels as the output[a]. For an input image $\mathbf{x}$, the output of the last convolutional block of the discriminator gives $f(\mathbf{x})$. For i-ACGAN-r, the $[\mathbf{y};\mathbf{z}]$ in (6.5) is updated using 5,000 iterations with a random initial value of $[\mathbf{y};\mathbf{z}]$. To

[a]We also tried 128×128 pixels for the output of the generator, but the synthetic images are more blurry. This is caused by the limits of the DCGAN architecture. With more sophisticated models, images of higher resolutions and better quality should be able to be generated. Nevertheless, the three methods described in this chapter are not restricted by a particular architecture.

avoid local optima, ten random initial values of $[\mathbf{y};\mathbf{z}]$ are used and the best inferred $[\mathbf{y};\mathbf{z}]$ that achieves the minimum $\|f(\mathbf{x}) - f(\mathcal{G}(\mathbf{y},\mathbf{z}))\|_2^2$ is kept. A similar strategy is applied to i-ACGAN-d. The value of M in Algorithm 6 is set to 3, which allows around 80% of the ground truth labels to be included in set C_M. Larger values of M may slightly enhance the class recovery accuracy, but it significantly increases the runtime. For all three methods, the dimension of $\mathbf{z}$ is 100.

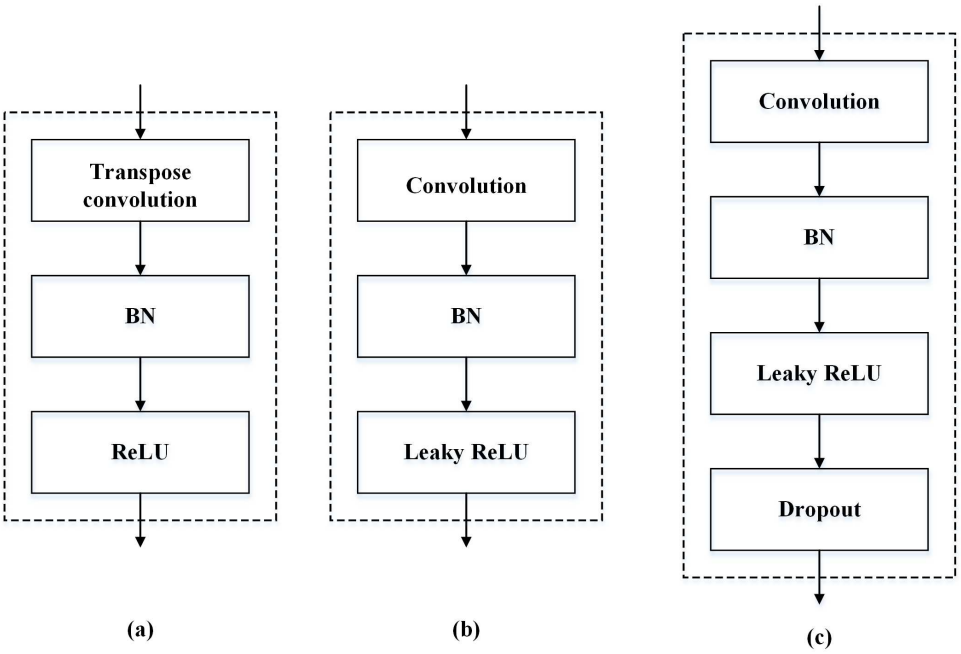


Figure 6.3: The convolutional blocks for (a) the generator, (b) the encoder, and (c) the discriminator of the ACGANs used in this chapter.

6.4 Experimental results

The three methods were tested with two performance measures on several natural scene datasets. This section begins with the introduction of the datasets and performance measures. Then, it proceeds to present and analyze the experimental results.

6.4.1 Datasets and performance measures

The datasets. The first dataset is the Intel Image Classification Dataset (Intel, for short)[b]. This dataset contains images of 150×150 pixels for six classes: 1) "buildings", 2) "forest", 3) "glacier", 4) "mountain", 5) "sea" and 6) "street". There are 14,034 images in its training set, with each class having around 2,000 training images. These training images were used to train the ACGANs, which were then tested on a test set containing 3,000 images. Some sample images from this dataset are shown in Figure 6.4.

Figure 6.4: Samples from the Intel Dataset. The top row shows training samples while the bottom row shows testing samples. From left to right, the classes are 1) "buildings", 2) "forest", 3) "glacier", 4) "mountain", 5) "sea" and 6) "street".

The second dataset, called "Places8" in this chapter, includes eight classes from the Places dataset [144]: 1) "fountain", 2) "highway", 3) "iceberg", 4) "ocean", 5) "river", 6) "train railway", 7) "volcano" and 8) "wind farm". Each class has around 15,000 training images of 256×256 pixels. Two test sets were constructed. The first test set contains 1,600 static images manually downloaded from the internet, with 200 images in each class. The images in this test set has a wide diversity in that none of them are similar to each other. This test set is named "Place8 Static". The second test set is comprised of 1,692 video frames sampled from 210 videos from two dynamic scene datasets, the Maryland dataset [25] and the Yupenn dataset [7]. In particular, the Maryland dataset contributed three classes each with ten videos: 1) "fountain", 2) "iceberg collapse" and 3) "volcano

[b] https://www.kaggle.com/puneet6060/intel-image-classification.

eruption". The Yupenn dataset contributed six classes each with 30 videos: 1) "fountain", 2) "highway", 3) "ocean", 4) "railway", 5) "rushing river" and 6) "windmill farm". From each video, seven to ten frames were regularly sampled depending on the length of the video. This test set is named "Place8 Dynamic" as it contains images from videos. Some sample images for Places8, Place8 Static and Place8 Dynamic are shown in Figure 6.5. Note that unlike Place8 Static, some images in Place8 Dynamic are similar if they are from the same video. These datasets are summarized in Table 6.2. To comply with the generator of the ACGANs, an image is downsized to 64×64 pixels before being fed into the discriminator or the encoder of an ACGAN.

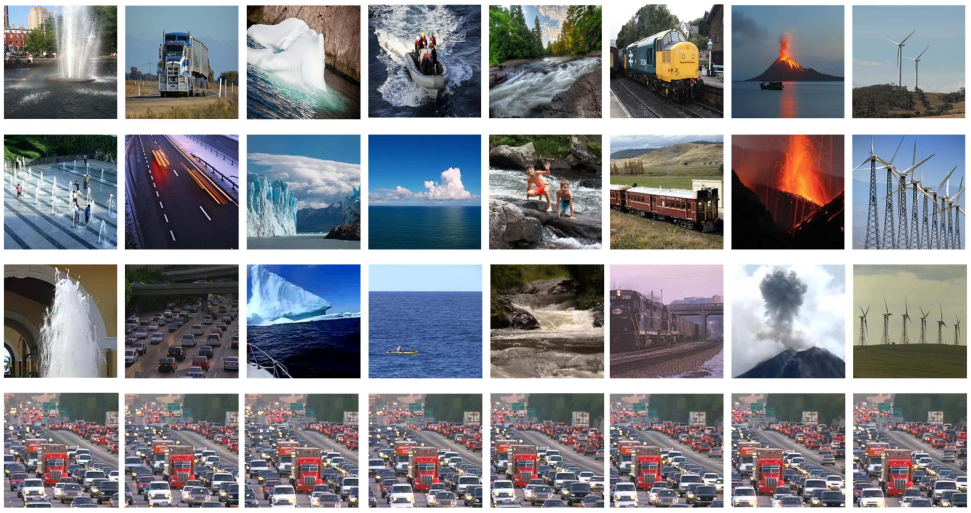


Figure 6.5: Samples from the Places8 and its two test sets. The first row shows samples from Places8, while the second and third rows show samples from Place8 Static and Place8 Dynamic, respectively. From left to right, the classes are 1) "fountain", 2) "highway", 3) "iceberg", 4) "ocean", 5) "river", 6) "train railway", 7) "volcano" and 8) "wind farm". The bottom row shows frames sampled from a "highway" video from Place8 Dynamic.

Table 6.2: Summary of the datasets used for training and testing the three methods.

Dataset	**Description**
Intel	Six classes, 14,034 training images (around 2,000 images per class), one test set of 3,000 images.
Places8	Eight classes, 117,257 training images (around 15,000 images per class), two test sets: Place8 Static (1,600 images) and Place8 Dynamic (1,692 images).

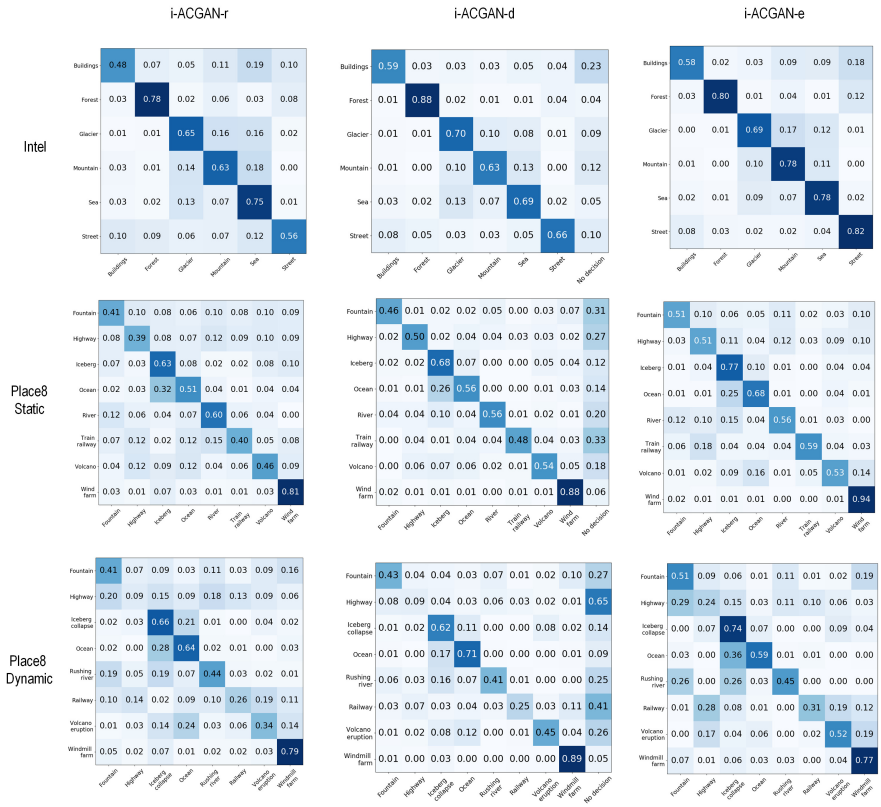


Figure 6.6: Confusion matrices to show the performance of the three methods on the three test sets.

Two performance measures. Existing criteria to measure the performance of a GAN, such as the Inception Score [177] and the Fréchet Inception Distance [178], aim to judge the quality of generated images and their diversity in a general way. These criteria are not suitable in this study, however, as the aim here is to measure the capability of an approach to infer the input to the generator. In this chapter, two performance measures are used: 1) class recovery accuracy and 2) image reconstruction error. The *class recovery accuracy* measures the accuracy rate of recovering the conditional category label from the inferred generator input,

$$Ac = \frac{N_{\text{correct}}}{N_{\text{all}}}, \tag{6.9}$$

where N_{correct} is the number of images with correctly inferred labels, and N_{all} is the total number of images in the test dataset. A high Ac indicates the approach is good at inferring the conditional class label. The *image reconstruction error*

measures the error between a real image $\mathbf{x}$ and the generated image $\tilde{\mathbf{x}}$. Here the pixel-wise mean-square error (MSE) is used for this purpose, given by $\frac{1}{N}\|\mathbf{x}-\tilde{\mathbf{x}}\|_2^2$, where N is the number of pixels in $\mathbf{x}$. The feature-element-wise MSE was also tested, given by $\frac{1}{N_f}\|f(\mathbf{x})-f(\tilde{\mathbf{x}})\|_2^2$, where N_f is the number of elements in $f(\mathbf{x})$. Note that the feature-element-wise MSE is different from Lucic *et al.*'s precision criterion [179], which measures the distances of generated images to an artificial data manifold.

6.4.2 Results and analysis

In this section, results of the three methods tested on the datasets are presented and analyzed in various respects.

(1) Class recovery accuracy comparison. Table 6.3 shows the *Ac* values obtained by the three methods. It can be seen from the table that i-ACGAN-r performs the worst and i-ACGAN-e performs the best. The performance of i-ACGAN-d lies between the two. Clearly, class labels are better inferred with the encoder than with inverting the generator of an ACGAN. i-ACGAN-d outperforms i-ACGAN-r because the latter treats a discrete-continuous optimization problem as a continuous one, and thus the sought $\mathbf{y}$ is continuous and suboptimal. By contrast, the former decomposes the optimization problem into multiple continuous sub-problems such that discrete labels can be inferred separately. From the same table, it can also be seen that Place8 Dynamic dataset is more challenging than the other two datasets. This may be attributed to the least diversity of the images in this dataset—if an image is misclassified, those images similar to it may also be misclassified.

To better reveal the performance of the three methods, confusion matrices for each of them are shown in Figure 6.6. Each row in a confusion matrix describes how the images in a given class are correctly labeled or mislabeled as other classes. Taking the first confusion matrix in this figure as an example, 48% of the

Table 6.3: Class recovery accuracy (in %) obtained by the three methods on the test datasets.

	Method		
Dataset	i-ACGAN-r	i-ACGAN-d	i-ACGAN-e
Intel	64.30	69.07	74.37
Place8 Static	52.50	58.25	63.43
Place8 Dynamic	44.56	47.28	49.88

test images in the "building" classes are correctly labeled, 7% of the images in this class are mislabeled as "forest", and so on. The values in a confusion matrix range between zero and one, and the sum of the elements in each row is one. An extra entry "no decision" is added into the confusion matrices for i-ACGAN-d, to reflect that no predictions are made if the ground truth label of a test image is not included in set C_M. It can be seen from these confusion matrices that i-ACGAN-e generally outperforms the other two methods in terms of having the highest correct labeling rates in most classes. For example, it achieves the highest correct labeling rates in seven out of eight classes (except "volcano") for Place8 Static dataset. For the most challenging Place8 Dynamic dataset, it achieves the highest correct labeling rates in six out of eight classes.

Finally, we found out that in order to maintain the *Ac* rate, minimizing the differences between the features of a generated image and a real image is better than directly minimizing the pixel-level differences between the two images. To support this argument, the term $\min_{\mathbf{y},\mathbf{z}} \|\mathbf{x} - \mathcal{G}(\mathbf{y},\mathbf{z})\|_2^2$ was used to replace $\min_{\mathbf{y},\mathbf{z}} \|f(\mathbf{x}) - f(\mathcal{G}(\mathbf{y},\mathbf{z}))\|_2^2$ (Eq. (6.4)) for i-ACGAN-r. This alternative was tested on a subset of 160 images from the Intel dataset. We observed a sharp decrease in the *Ac* value from 64.38% to 39.38%. This is because the features of the images are more discriminative than the images themselves.

(2) Class recovery accuracy comparison at the video level. As Place8 Dynamic consists of video frames, a class recovery accuracy can be computed at the video level: each video frame in this test set is assigned an image-level label, and the

video-level label is predicted using majority voting from the image-level labels. Table 6.4 shows the class recovery accuracies obtained at the video level by the three methods.

Table 6.4: Video-level class recovery accuracy (in %) obtained by the three methods on Place8 Dynamic dataset.

	Method		
Place8 Dynamic	i-ACGAN-r	i-ACGAN-d	i-ACGAN-e
Video-level class recovery accuracy	54.29	63.33	50.00

Surprisingly, i-ACGAN-e performs worse than the other two methods in this test set. To reveal the reasons behind this phenomenon, we computed the information entropy from the distributions of the image-level labels. The information entropy measures the extent of disorder in a system. The higher the entropy value, the higher extent of disorder in the system. For a given video that has K frames, in which K_1 frames are correctly labeled while K_0 frames are mislabeled, its entropy is defined as:

$$E = -\sum_{j=0}^{1} p_j \ln p_j, \tag{6.10}$$

where $p_0 = K_0/K$ and $p_1 = K_1/K$. For instance, if $p_0 = 0$ and $p_1 = 1$, then $E = 0$. As another example, if $p_0 = p_1 = 0.5$, then $E = 0.69$, which indicates a higher extent of disorder in predicting the image-level labels for this video. Table 6.5 shows the average entropy for all the videos computed using the three methods. It can be seen from this table that i-ACGAN-e records the smallest average entropy. This fact suggests that although i-ACGAN-e obtains the highest class recovery accuracy at the image level, the distributions of the correct labels and incorrect labels of a video are more "central" than the other two methods. However, this centralness may not necessarily lead to better video-level class recovery accuracy. Consider an illustrative example of two videos. Assume that i-ACGAN-e obtains 50% in the image-level class recovery accuracy while i-ACGAN-r obtains just

40%. Moreover, in the case of i-ACGAN-e, assume that all the frames for one video are correctly labeled while for the other video all the frames are mislabeled. In this case, the video-level class recovery accuracy obtained by i-ACGAN-e is 50%. Now let us assume that 40% of frames in both videos are correctly labeled by i-ACGAN-r; however, these correct labels happen to be the majority labels. In that case, the video-level class recovery accuracy obtained by i-ACGAN-r is 100%.

Table 6.5: Average entropy for all the videos computed using the three methods on Place8 Dynamic dataset.

	Method		
Place8 Dynamic	i-ACGAN-r	i-ACGAN-d	i-ACGAN-e
Average entropy (sum entropy/number of videos)	0.40	0.31	0.18

(3) Image reconstruction error comparison. The image reconstruction errors (the pixel-wise MSE) produced by the three methods are shown in Table 6.6. Here, the errors are computed only for the correctly predicted images because the three methods have different *Ac* rates. It can be seen from the table that the image reconstruction errors by i-ACGAN-r and i-ACGAN-d are close. This is because both methods use a similar strategy to solve for the latent sample $\mathbf{z}$. However, the image reconstruction errors by i-ACGAN-e are much higher than those by the other two methods. This can also be perceptible by visually inspecting the images synthesized by these methods, some of them are shown in Figure 6.7.

Table 6.6: Pixel-wise MSE (mean±std) produced by the three methods on the three test sets.

	Method		
Dataset	i-ACGAN-r	i-ACGAN-d	i-ACGAN-e
Intel	1650.69 ± 1130.56	1792.78±1261.39	4407.08±2096.52
Place8 Static	1636.05±1174.05	1533.04±1371.61	3444.81±2292.99
Place8 Dynamic	1111.26±876.67	1127.37±874.89	3799.21±2187.19

Figure 6.7: Samples of real and synthetic images of the three test sets.

Surprisingly, the pixel-wise MSEs of the three methods are not consistent with their feature-element-wise MSEs. Table 6.7 shows the feature-element-wise MSEs produced by the three methods. As can be seen from this table, i-ACGAN-e produces the minimal errors at the feature level among the three methods. A comparison of Table 6.6 and Table 6.7 suggests that the feature-element-wise MSE deviates from the pixel-wise MSE with the introduction of the encoder. This may be caused by the complex interactions between the discriminator and the encoder in i-ACGAN-e. To further reveal the performance of i-ACGAN-e, we re-trained it by replacing $\lambda\|f(\mathbf{x})-f(\mathcal{G}(\mathcal{E}(\mathbf{x})))\|_2^2$ in Eq. (6.8) with $\lambda\|\mathbf{x}-\mathcal{G}(\mathcal{E}(\mathbf{x}))\|_2^2$. When tested on the Intel dataset, the pixel-wise MSE increased to 5053.06±2394.19, while the class recovery accuracy decreased to 69.63%. This shows that the feature similarity works better than the pixel similarity between the synthesized image and the real image for i-ACGAN-e.

(4) Runtime comparison. All the three methods were tested on a workstation equipped with an NVIDIA GeForce GTX 1080Ti single GPU and Ubuntu 16.04 operating system. The time for updating $\mathbf{y}$ and $\mathbf{z}$ (and $\mathbf{z}$ alone for i-ACGAN-d)

Table 6.7: Feature-element-wise MSE (mean±std) produced by the three methods on the three test datasets.

Dataset	Method		
	i-ACGAN-r	i-ACGAN-d	i-ACGAN-e
Intel	0.02600±0.02679	0.02637±0.02308	0.00291±0.00248
Place8 Static	0.02604±0.03539	0.02390±0.02717	0.00130±0.00124
Place8 Dynamic	0.01650±0.01541	0.01605±0.01325	0.00101±0.00078

once is around four milliseconds. It is in the same order of time required for directly inferring **y** and **z** with i-ACGAN-e. However, i-ACGAN-e is much faster than the other two methods as the inferring is done once and for all. By contrast, i-ACGAN-r and i-ACGAN-d need to update **y** and **z** (and **z** alone for i-ACGAN-d) for many iterations. It took these two methods several weeks to test the three datasets. i-ACGAN-d was about three times slower than i-ACGAN-r. This is because the complexity of i-ACGAN-d is about three times higher than that of i-ACGAN-r. By contrast, it only took i-ACGAN-e several minutes to do the same job.

(5) Influence of λ in i-ACGAN-e. We experimented the influence of λ values on the two performance measures for i-ACGAN-e. Figure 6.8 shows the results of different values of λ tested on the Intel dataset. It can be seen from the figure that the Ac values fluctuate with different values of λ. However, the decreasing trend of the image reconstruction errors is more obvious with increasing values of λ. When $\lambda > 10$, the image reconstruction errors tend to become flat. Thus, a $\lambda = 10$ was adopted as a compromise between the two performance measures.

6.5 Chapter summary

ACGANs have the potential to be used as a reconstruction-based classifier. To infer the input to the generator of an ACGAN, three methods are designed in this chapter, namely i-ACGAN-r, i-ACGAN-d, and i-ACGAN-e. The methods are

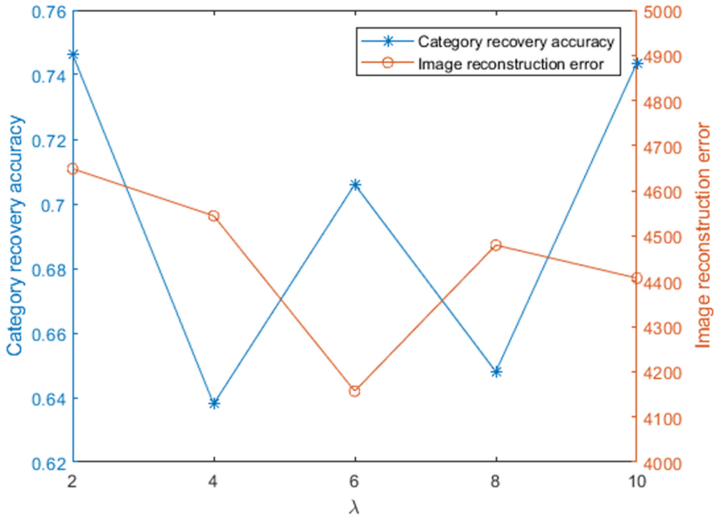


Figure 6.8: Influence of different values of λ in i-ACGAN-e on the two performance measures.

compared with respect to two performance measures, the class recovery accuracy and the image reconstruction error, on several natural scene datasets. Experimental results show that i-ACGAN-e outperforms the other two methods in terms of the class recovery accuracy criterion and runtime. However, the images generated by i-ACGAN-r and i-ACGAN-d have smaller image reconstruction errors than i-ACGAN-e.

Chapter 7

Stereo Matching: A Preliminary Step for 3D Dynamic Scene Recognition

Chapter contents

This chapter is based on the following publication:

X. Peng, A. Bouzerdoum, S. L. Phung, "Efficient cost aggregation for feature-vector-based wide-baseline stereo matching," EURASIP Journal on Image and Video Processing, doi:10.1186/s13640-018-0249-y, 2018.

For a better understanding of a complex dynamic scene, information sourced from multiple cameras can be more useful than that from a single camera. One advantage of the multi-camera setup over a single camera is that depth information about the dynamic scene can be inferred from the images output by the multiple cameras. With the depth information, the 3D shape of the scene can be reconstructed so that 3D dynamice scene recognition can be carried out. One preliminary step to estimate the depth from multiple cameras is stereo matching. In stereo matching applications, local cost aggregation techniques are usually preferred over global methods due to their speed and ease of implementation. Local methods make implicit smoothness assumptions by aggregating costs within a finite window; however, cost aggregation is a time-consuming process. Furthermore, most existing local methods are based on pixel intensity values, and hence are not efficient with feature vectors used in wide-baseline stereo matching. In this chapter, a new cost aggregation method is proposed, where a per-column cost (PCC) matrix is combined with a feature-vector-based weighting strategy to achieve both matching accuracy and computational efficiency. Here, the proposed cost aggregation method is applied with the DAISY feature descriptor for wide-baseline stereo matching; however, other features such as deep-learned features may also be used with this method. A performance comparison with several benchmark local cost aggregation approaches is presented, along with a thorough analysis of the time and storage complexity of the proposed method.

7.1 Introduction

Automatic understanding a complex outdoor dynamic scene is pivotal to some applications, one of which is self-driving. One preliminary sub-task of this prob-

lem is depth estimation. Once the depth information is available, a 3D model of the scene can be reconstructed and used for subsequent sub-tasks such as scene categorization, object detection and tracking, and event categorization. One illustrative example is shown in Figure 7.1. Currently, laser sensors are mainly used for this purpose. They are highly accurate, but also bulky, expensive and energy-inefficient. An alternative is to use inexpensive stereo cameras to estimate the depth information. Estimating depth from a pair of stereo images is a long-standing problem in computer vision. Its aim is to find a dense correspondence map between a pair of stereo images to generate either a disparity map (for rectified stereo pairs), or a depth map (for known camera calibration parameters). Stereo algorithms generally involve the following four steps: 1) matching cost computation, 2) cost aggregation, 3) disparity computation or optimization, and 4) disparity refinement [180]. Both *local* and *global* approaches need to perform the matching cost computation step, but they differ in the treatment of smoothness constraints. Local methods make implicit smoothness assumptions by aggregating costs within a finite window. Global approaches, by contrast, make explicit smoothness assumptions by combining the data and the smoothness terms into a cost function, which is subsequently optimized using an iterative procedure. The most commonly used optimization methods for global approaches include expectation-maximum (EM) [181], cooperative optimization [182, 183], graph cuts (GC) [184], max-product loopy belief propagation (MPLBP) [185], and tree-reweighted message passing (TRW) [186]. The last three methods are categorized as energy minimization for Markov Random Fields (MRFs) [187]. In practical applications, local approaches are preferred to their global counterparts due to their speed and ease of implementation.

The focus of this chapter is local methods. In particular, a local cost aggregation method is proposed that operates on feature vectors. The proposed method has two major contributions. First, a feature-vector-based weighting strategy is developed, which can be computed much more efficiently than conventional

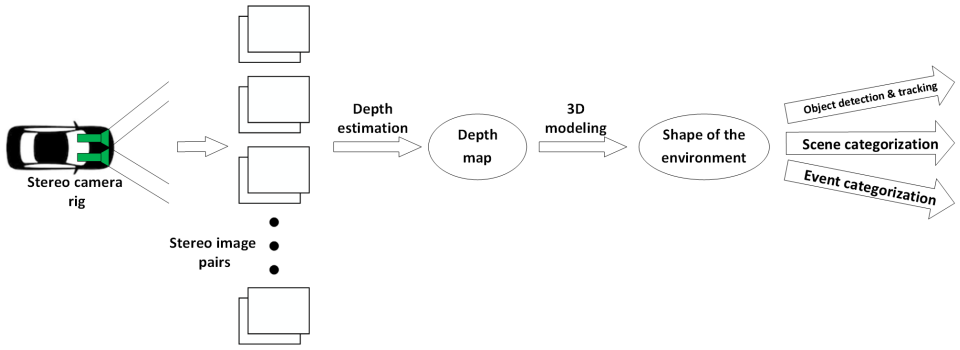


Figure 7.1: An illustrative example of understanding a complex outdoor dynamic scene for self-driving. In this example, the stereo camera rig provides stereo video sequences using which the depth of the scene is estimated. A 3D model of the scene can then be reconstructed, which is useful for subsequent sub-tasks.

bilateral filtering, but with only a slight reduction in accuracy. Second, a new concept called the PCC matrix is proposed to share the aggregated costs across different disparities during cost aggregation. This is in sharp contrast to other cost aggregation strategies, such as Integral Images [188] and box-filtering [189], which are limited to a single disparity map. Although the DAISY feature [190] is used to instantiate the proposed method, other feature vectors can also be applied.

The rest of this chapter is organized as follows. In Section 7.2, the related work is discussed. In Section 7.3, the cost aggregation problem is formulated under the filtering framework, followed by the delineation of the proposed method. In Section 7.4, experimental results are presented, followed by a comprehensive analysis and discussion of the results. Finally, Section 7.5 summarizes this chapter.

7.2 Related work

Existing short-baseline stereo matching methods, which are mostly based on pixel intensity values, perform reasonably well and are quite fast. Di Stefano *et al*. proposed a local method which achieved real-time speed using Single Instruction Multiple Data (SIMD) implementation [189]. Tombari *et al*. compared fourteen cost aggregation techniques in terms of accuracy and computation cost [191]. They found that cost aggregation methods using adaptive weights are among the

most accurate. Hirschmüller proposed a semi-global method based on mutual information [192]. Based on the gestalt principles, Yoon and Kweon developed an edge-preserving bilateral filter for stereo matching [193]. Subsequently, Mattoccia *et al*. proposed a symmetric adaptive weighting strategy using two independent spatial and range filters [194]. Instead of adopting an exact weighting strategy, Min *et al*. addressed the cost aggregation issue by introducing two approximations [195]. In another approach, Hosni *et al*. formulated the stereo matching problem in a cost-volume filtering manner [196]. The cost volume is a 3D array that stores the costs for choosing a label (*i.e*. the disparity value in the stereo matching process) at a given pixel. To maintain boundaries in the filtered output of the guidance image (which is the left image of a stereo pair), the filter weights are chosen to be those of the guided filter [197]. Yang developed a Minimum-Spanning-Tree-based cost aggregation method, which avoids the local optimality caused by manually specifying the size of the supporting window [198]. Inspired by their work, Mei *et al*. introduced a segment-tree-based cost aggregation method [199]. Zhang *et al*. showed that the different cost aggregation methods essentially differ in the choice of similarity kernels and can be reformulated in a unified optimization framework [200].

Matching measures directly constructed using pixel intensity values, such as sum of absolute differences (SAD), sum of squared differences (SSD), and normalized cross correlation (NCC), lack robustness to large perspective distortions. These measures are not suitable for wide-baseline stereo matching, where there are significant variations in the viewpoints. A better alternative to pixel-intensity-based cost aggregation is to employ local feature descriptors. In recent years, many new feature detectors and descriptors have been developed, including BRIEF [201], BRISK [202], FREAK [203] and ORB [204]. At the same time, several studies analyzed the performance of feature vectors on different tasks. Heinly *et al*. compared the performance of five descriptors, BRIEF, BRISK and ORB, SIFT [22], and SURF [205] in feature detection and description [206]. Khan

et al. used precision-recall curves and the Wilcoxon signed rank test to compare the performance of thirteen feature vectors [207]. They found the SIFT is the most accurate performer in both image recognition and feature matching applications. The increasingly abundant feature descriptors provide alternatives to pixel-intensity-based similarity measures in dense matching scenarios, as shown in [190] and [146]. Tola *et al*. showed that the DAISY feature descriptor, SIFT and SURF all outperform the NCC and pixel difference in wide-baseline stereo matching [190]. To accelerate the cost aggregation step using the BRIEF feature descriptor, Zhang *et al*. incorporated binary masks into the cost aggregation term [208]. The binary masks are constructed using the SAD measure between two pixels in the CIELAB color space. However, their binary masks can only be paired with binary feature vectors like BRIEF. Thus, this method is not applicable to general real-valued feature vectors such as SIFT and DAISY.

Stereo matching using feature vectors has two difficulties. First, feature-vector-based matching costs are much more computationally intensive than pixel-intensity-based ones. There are several pixel-intensity-based strategies for reducing cost aggregation [194, 195, 196, 197]. However, they do not work well when directly applied to feature vectors. For example, the computational load of the similarity kernels in the bilateral filtering methods [193, 194] and the tree-based methods [198, 199] is quite low when involving only pixel-wise intensity differences. However, their computational load is very high if feature-vector-based similarity measures are used. Second, storing feature vectors for each image pixel requires a large amount of memory. To facilitate the repetitive testing of different pixel correspondences, it is desirable to store the per-pixel feature vectors, as done in SIFT flow [146]. As an example of storage cost, to store a 200-element DAISY feature vector in double-precision floating-point format for each pixel of a stereo pair of 1920×1080 pixels, approximately a storage of $(2 \times 1920 \times 1080 \times 8 \times 200)/1024^3 \approx 6.18$ GB is needed. Obviously, this is not practical on memory-limited systems.

Recently, deep learning has been used to compare image pairs for stereo matching [209, 210, 211, 212, 213, 214, 215, 216]. In these methods, a CNN is deployed to compute the matching cost between a pair of image patches from the left and right images. Zagoruyko and Komodakis extracted feature descriptors from image patches at the branches of their Siamese network [209]. Their feature descriptor can be used as an alternative to hand-crafted feature descriptors, such as SIFT and DAISY. Žbontar and LeCun developped a CNN that directly outputs the matching cost between a pair of image patches [210]. The matching cost is then combined with a cross-based cost aggregation method. Chen *et al*. proposed a multi-scale deep embedding model to extract features from a pair of image patches [211]. The inner product of the features is large if the pair of image patches matches well, and vice versa. Luo *et al*. adopted a four-layer Siamese architecture for their CNN [212]. However, they observed that simply predicting the most likely configuration for every pixel using only the CNN output is not competitive with other modern stereo algorithms. To achieve a better performance, they combined their CNN with semi-global block matching and sophisticated post-processing. Several methods adopt 2D encode-decoder architectures [213, 214, 215]. The iResNet [213] incorporates all the four steps involved in stereo matching into one network, so that all steps share the same features and are optimized jointly. The SegStereo [214] integrates semantic cues from segmentation into disparity estimation. Zhai *et al*. proposed the so-called dilated residual networks to learn the optical flow between two images [215]. Their architecture avoids the loss of details that are common with the U-Net architecture. The EdgeStereo [216] is a multi-task architecture, which is composed of a disparity estimation branch and an edge detection branch. All these deep-learning methods rely on the availability of a large pool of annotated pairs of image patches to learn a mapping between them. To address this issue, Mayer *et al*. established a synthetic dataset containing 35,000 stereo image pairs with ground truth disparity, optical flow, and scene flow [217]. Though deep-learning-based methods are current state-of-

the-art, the proposed method has an advantage in that it does not require any training data. Therefore, it can be easily deployed to practical applications.

7.3 Cost aggregation method using feature vectors

In this section, first, cost aggregation is cast as a filtering problem. Then, an analysis is made to show why existing pixel-intensity-based cost aggregation methods are not suitable for feature vectors. Finally, the proposed method is described in detail.

7.3.1 Cost aggregation using a filtering framework

The proposed method works on a pair of *rectified* stereo images, where the search of corresponding points on the stereo image pair is constrained on the horizontal image scanlines. Consider a pair of *left* image I_l and *right* image I_r. The aim is to find a pixel $q = (x + d, y)$ in I_r which corresponds to a pixel $p = (x, y)$ in I_l, where d is the *disparity* between the pixel pair, $d \in \mathcal{D} := [d_{\min}, d_{\max}]$. Let $I_l(x, y)$ denote the intensity value (or the vector of color values) at location (x, y) in image I_l. The search for pixel q is carried out along a horizontal scan line. For a chosen feature vector $\mathbf{f}$ of length L, the dissimilarity between pixels p and q is computed by comparing $\mathbf{f}(I_l; p)$ and $\mathbf{f}(I_r; q)$, which denote the feature vectors extracted at pixels p and q in I_l and I_r, respectively. Here, no restrictions are applied on the type of feature vectors that can be used. For simplicity and without loss of generality, the images I_l and I_r have identical sizes (H rows, W columns). Furthermore, the search range $\mathcal{D}$ has M discrete integer values, and is the same for all the pixels p in I_l. The dissimilarity between two feature vectors $\mathbf{f}_1$ and $\mathbf{f}_2$ is denoted as $c(\mathbf{f}_1, \mathbf{f}_2)$.

Cost aggregation can be defined as a filtering process [200]. Let $\mathbf{C}$ denote the cost volume of size $W \times H \times M$, where $C(x, y, d)$ stores the cost for pixel (x, y) at disparity level d. Let $\nabla_x I$ denote the gradient component of image I along the x direction. For a pixel p in image I_l, the combined intensity and gradient cost

volume, used in [196, 198, 199], is defined as

$$C(x,y,d) = (1-\alpha)\min(\|I_l(x,y) - I_r(x+d,y)\|_2, \tau_1) + \alpha\min(\|\nabla_x I_l(x,y) - \nabla_x I_r(x+d,y)\|_2, \tau_2), \tag{7.1}$$

where $1 \le x \le W$, $1 \le y \le H$, and $d_{\min} \le d \le d_{\max}$. The parameters τ_1 and τ_2 are two thresholds, and α balances the color and gradient terms. Consider a 2D filter $\mathbf{K}$ (also called the "similarity kernel" in [200]). For a supporting window of size $(2w+1)\times(2w+1)$ and centered at pixel (x,y) in the left image I_l, the filter output can be expressed as

$$\tilde{C}(x,y,d) = \sum_{i=-w}^{w}\sum_{j=-w}^{w} K(i,j)\, C(x+i,y+j,d). \tag{7.2}$$

Existing cost aggregation methods differ mainly in the choice of $\mathbf{K}$. For example, for the edge-preserving bilateral filter [193], the kernel is given as

$$K^{\text{bf}}(i,j) = \frac{1}{Z}\exp\Big(-\frac{\sqrt{i^2+j^2}}{\sigma_s}\Big)\exp\Big(-\frac{\|I_l(x,y) - I_l(x+i,y+j)\|_2}{\sigma_c}\Big), \tag{7.3}$$

where $-w \le i,j \le w$, the parameters σ_s and σ_c control the spatial and color similarity, and Z is a normalization constant such that $\sum_{i=-w}^{w}\sum_{j=-w}^{w} K^{\text{bf}}(i,j) = 1$. In the case of the guided-image filter [197], the kernel is given as

$$K^{\text{gf}}(i,j) = \frac{1}{(2w+1)^2}\,[1 + (I_l(x,y) - \mathbf{u})^T(\mathbf{\Sigma} + \epsilon\mathbf{I})(I_l(x+i,y+j) - \mathbf{u})], \tag{7.4}$$

where $\mathbf{u}$ is the mean vector, $\mathbf{\Sigma}$ is the covariance matrix of the pixels in the supporting window, $\mathbf{I}$ denotes the identity matrix, and ϵ is a smoothness parameter. Note that for both filters, the filter kernel varies according to the coordinates of the center pixel (x,y). In the tree-based cost aggregation methods [198, 199], a connected, undirected graph $\mathcal{G} = (\mathcal{V},\mathcal{E})$ is established for I_l, where the set of vertices $\mathcal{V}$ is the image pixels and the edges $\mathcal{E}$ connect neighboring pixels. In all these kernels,

the weight between two pixels is defined as the difference between their intensity values.

One important reason that the state-of-the-art cost aggregation methods can work very fast is that the similarity kernels can be computed very efficiently, with simple arithmetic operations mostly. However, this is not the case when feature vectors rather than pixel intensity values are used. For example, if the pixel difference term $\|I_l(x,y) - I_l(x+i,y+j)\|_2$ in Eq. (7.3) is replaced with the feature-vector-based dissimilarity term $c(\mathbf{f}(I_l;x,y),\mathbf{f}(I_l;x+i,y+j))$, a sharp increase in the computation will be incurred.

7.3.2 Feature-vector-based cost aggregation

To formulate the feature-vector-based cost aggregation problem as a filtering problem, the cost volume $\mathbf{C}$ needs to be computed first:

$$C(x,y,d) = c(\mathbf{f}(I_l;x,y),\mathbf{f}(I_r;x+d,y)), \tag{7.5}$$

where $x \in [1,W]$, $y \in [1,H]$, and $d \in [d_{\min},d_{\max}]$.

The dissimilarity measure $c(\mathbf{f}(I_l;x,y),\mathbf{f}(I_r;x+d,y))$ is problem-dependent. For example, a commonly used dissimilarity measure is the Euclidean norm of the difference between the two feature vectors. To avoid repeatedly computing and comparing feature vectors, the cost $C(x,y,d)$ is computed in a row-wise manner. For each row y, two $L \times W$ matrices are formed:

$$\begin{aligned}\mathbf{F}_l^y &= [\mathbf{f}(I_l;1,y),\mathbf{f}(I_l;2,y),\ldots,\mathbf{f}(I_l;x,y),\ldots,\mathbf{f}(I_l;W,y)], \\ \mathbf{F}_r^y &= [\mathbf{f}(I_r;1,y),\mathbf{f}(I_r;2,y),\ldots,\mathbf{f}(I_r;x,y),\ldots,\mathbf{f}(I_r;W,y)].\end{aligned} \tag{7.6}$$

The x-th column of $\mathbf{F}_l^y$ contains the feature vector computed at pixel (x,y) in the left image I_l. The x-th column of $\mathbf{F}_r^y$ contains the feature vector computed at pixel (x,y) in the right image I_r. To compute $c(\mathbf{f}(I_l;x,y),\mathbf{f}(I_r;x+d,y))$, the feature vectors

$\mathbf{f}(I_l; x, y)$ and $\mathbf{f}(I_r; x+d, y)$ are retrieved directly from $\mathbf{F}_l^y$ and $\mathbf{F}_r^y$ rather than being re-computed. The next subsection presents in detail the proposed method for cost aggregation.

7.3.2.1 Feature-vector-based weighting strategy

The proposed feature-vector-based weighting strategy is inspired by the edge-preserving bilateral filter [193], but is modified to accommodate feature-vector-based dissimilarity measures. Figure 7.2 shows the difference between the conventional bilateral filter and the proposed bilateral filter. In the conventional bilateral filter, the weights $K^{\text{bf}}(i, j)$ of Eq. (7.3) are computed with a single center pixel as the reference. However, in the proposed bilateral filter, the weights are computed with multiple center pixels as the reference.

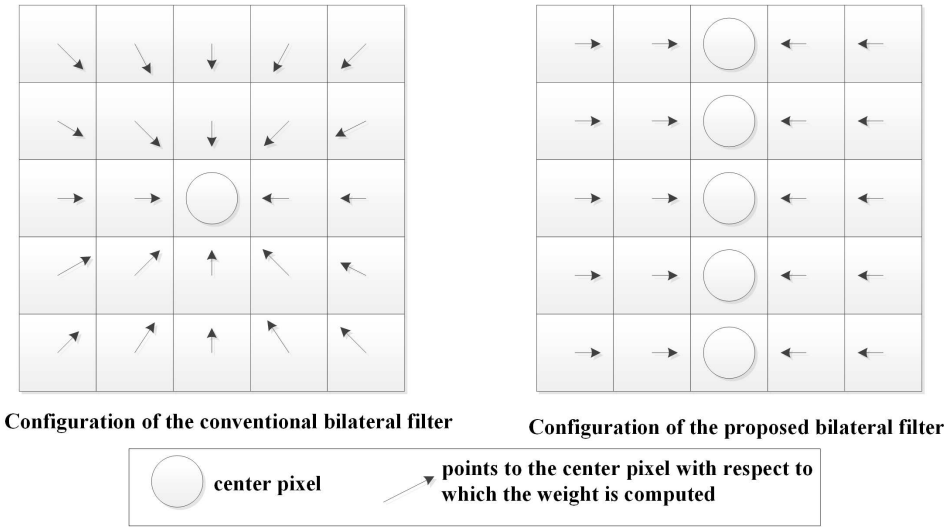


Figure 7.2: Difference between the conventional bilateral filter and the proposed bilateral filter, where arrows point to the "center" pixels (denoted as circles). Left: weights for the conventional bilateral filter are computed with one single "center" pixel as the reference. Right: weights of the proposed bilateral filter are computed with multiple "center" pixels as the reference.

Consider a supporting window of size $(2w+1)\times(2w+1)$ located at pixel (x, y) in an image. The weights of the proposed bilateral filter are defined as

$$G^{\text{bf}}(i, j) = \exp\Big(-\frac{|i|}{\sigma_1}\Big)\exp\Big(-\frac{c(\mathbf{f}(I_l; x, y+j), \mathbf{f}(I_l; x+i, y+j))}{\sigma_2}\Big), \tag{7.7}$$

where $-w \leq i, j \leq w$, and parameters σ_1 and σ_2 control the spatial and color similarity. From Eq. (7.7), if $i = 0$, $G^{\text{bf}}(i, j) = 1$. In other words, all the pixels in the middle column of the supporting window share the same weights, and act as "center" or "reference" pixels. As in the case of $K^{\text{bf}}(i, j)$, the weights of $G^{\text{bf}}(i, j)$ are normalized so that $\sum_{i,j} G^{\text{bf}}(i, j) = 1$.

The rationale of the proposed bilateral filter can be explained as follows. Because the feature vector $c(\mathbf{f}(I_l; x, y))$ describes a local area around a pixel (x, y) instead of just the pixel itself, the cost $c(\mathbf{f}(I_l; x, y), \mathbf{f}(I_l; x + i, y + j))$ is less spatially sensitive than $\|I_l(x, y) - I_l(x + i, y + j)\|_2$. This property enables us to use multiple center pixels for the proposed bilateral filter instead of a single center pixel as for the conventional bilateral filter. Furthermore, Eq. (7.7) is computed along the horizontal scan lines. If a single center is used, the term $c(\mathbf{f}(I_l; x, y), \mathbf{f}(I_l; x + i, y + j))$ needs to be computed *across* the scan lines, leading to a significant increase in the storage requirement. As confirmed later in Section 7.4.4, the proposed multi-center bilateral filter operates more efficiently while achieving a similar stereo matching accuracy compared with the single-center case.

7.3.2.2 Per-column cost matrix

The proposed weighting strategy works efficiently when combined with a new concept called the *per-column cost* (PCC) matrix. To illustrate this concept, an example is shown in Figure 7.3 of three supporting windows of size 3×3 pixels (i.e., $w = 1$) centered at locations (x, y), $(x + 1, y)$ and $(x + 2, y)$ in the left image. The three supporting windows share a common column (colored with red). For any of these three supporting windows, the counterparts of this red column in the d-shifted supporting windows in the right image occupy the same consecutive region of $(2w + 1) \times M$ pixels (shaded with gray). This fact implies that if the accumulated costs associated with the red column is recorded when matching either of the three supporting windows in the right image, the accumulated costs can be reused to reduce the computational load by a factor of $(2w + 1)$. This

observation leads to the construction of a matrix $\mathbf{\Gamma}_y$ to store the column-based accumulated costs:

$$\begin{aligned} \Gamma_y(x+d,x) &= \sum_{j=-w}^{w} c(\mathbf{f}(I_l;x,y+j),\mathbf{f}(I_r;x+d,y+j)) \\ &= \sum_{j=-w}^{w} C(x,y+j,d), \end{aligned} \tag{7.8}$$

where the subscript y indicates that it is constructed for the y-th row.

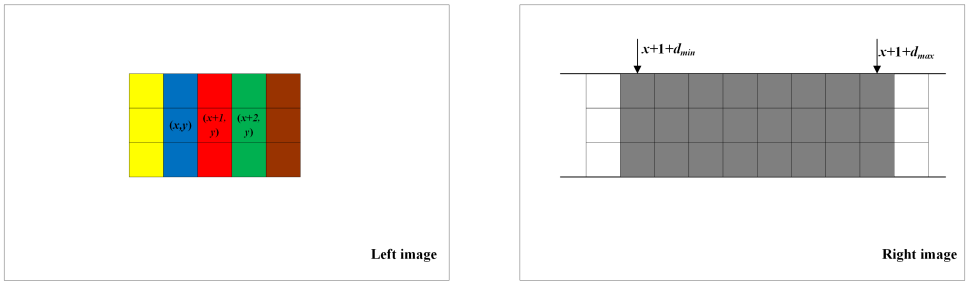


Figure 7.3: An example showing the 3×3 supporting windows ($w=1$) of three consecutive pixels (x,y), $(x+1,y)$ and $(x+2,y)$ in the left image. These three supporting windows share a common column (colored with red). For a given supporting window in the left image, a series of d-shifted supporting windows in the right image are matched against. The counterparts of this red column in these d-shifted supporting windows occupy an identical region of pixels in the right image (shaded with gray).

The matrix $\mathbf{\Gamma}_y$, of size $W\times W$, can be computed quite efficiently. First, it is quite sparse: for the x-th row of $\mathbf{\Gamma}_y$, all entries except those with indices from $(x+d_{\min})$ to $(x+d_{\max})$ are empty. Second, starting with $\mathbf{\Gamma}_{w+1}$, the matrix $\mathbf{\Gamma}_y$ for $y>w+1$ can be computed recursively as follows:

$$\Gamma_y(i,j) = \Gamma_{y-1}(i,j) + C(j,y+w,i-j) - C(j,y-1-w,i-j). \tag{7.9}$$

An important property of $\mathbf{\Gamma}_y$ is that it is shared across different disparities, as opposed to Integral Images and box-filtering, which are limited to a single disparity level.

Next, the weighting strategy described in Section 7.3.2.1 can be integrated in. From Eq. (7.8), each element of $\mathbf{\Gamma}_y$ accumulates the unweighted costs from

one column of the supporting window. However, the weights in Eq. (7.7) are computed pixel-wise. To address this incompatibility, the weights within one column of the supporting window are averaged. This averaged weight is used as a common weight for the accumulated costs within one column of the supporting window. This approximation is explained as follows: in Eq. (7.2), the weights $K(i,j)$ are normalized so that they sum to one for a given supporting window. To simplify the discussion, we rewrite Eq. (7.2) as

$$\tilde{C} = \sum_{s=1}^{2w+1} \sum_{t=1}^{2w+1} K_{s,t}\, C_{s,t} = \sum_{n=1}^{(2w+1)^2} K_n\, C_n, \tag{7.10}$$

where s is the column index, t is the row index, and n is the linear index of the pair (s,t). Here, C_n is the cost, and K_n is the positive weight associated with the n-th element in the supporting window, $\sum_n K_n = 1$. Now the elements in the supporting window can be grouped into $(2w+1)$ columns, the column-wise common weights (described above) approximate Eq. (7.10) by

$$\tilde{C} \approx \sum_{s=1}^{2w+1} \sum_{t=1}^{2w+1} \bar{K}_s\, C_{s,t} = \sum_{s=1}^{2w+1} \bar{K}_s \sum_{t=1}^{2w+1} C_{s,t}, \tag{7.11}$$

where the common weight for column s is denoted as $\bar{K}_s = \frac{1}{2w+1} \sum_{t=1}^{2w+1} K_{s,t}$. Note that $\sum_{s=1}^{2w+1} (2w+1) \bar{K}_s = \sum_{s=1}^{2w+1} \sum_{t=1}^{2w+1} K_{s,t} = 1$.

Now that $\sum_{t=1}^{2w+1} K_{s,t}$ represents the accumulated weights within one column of the supporting window, it can be computed in a way similar to the computation of the PCC matrix $\mathbf{\Gamma}$. To this end, a *per-column-weight* matrix $\mathbf{\Phi}$ of size $W \times W$ is initialized. To facilitate its computation, a 3D *per-pixel-weight* array $\mathbf{P}$ of size $W \times H \times (2w+1)$ is introduced. The pseudo-code for the proposed feature-vector-based cost aggregation algorithm is given in Algorithm 7. In the pseudo-code, the symbols $\mathbf{F}_l^y$, $\mathbf{F}_r^y$, $\mathbf{\Phi}_y$ and $\mathbf{\Gamma}_y$ explicitly emphasize that the corresponding computations are made along the y-th scanline. For simplicity, the "border problem" is not considered here. However, this restriction can be lifted by replicating the

border pixels in an appropriate way. The major steps of Algorithm 7 are also summarized in Figure 7.4.

An estimated disparity d is considered reliable if its aggregated cost $a(d)$ is smaller than $(2w+1)^2\,\tau$, where τ is a predefined threshold. Otherwise, we regard the pixel as occluded. A large aggregated cost indicates the feature vectors are not well matched, usually because the local regions lack distinctive visual appearance.

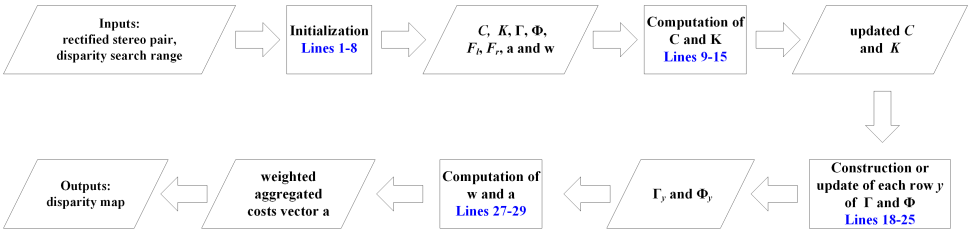


Figure 7.4: Block diagram of the proposed method.

Table 7.1: Time and storage complexity of the proposed feature-vector-based cost aggregation algorithm.

Time complexity	
Feature vector computation	$O_a(2\times W\times H)$
Feature vector comparison	$O_b(W\times H\times M)$ for C and $O_b(W\times H\times(2w+1))$ for P
Construction and update of $\mathbf{\Gamma}$	$O_c((2w+1)^2\times W\times M)$ and $O_c(3\times W\times H\times M)$, respectively
Construction and update of $\mathbf{\Phi}$	$O_c(\frac{(2w+1)^2}{2}\times W)$ and $O_c(3/2\times W\times H\times(2w+1))$, respectively
Cost aggregation for all pixels	$O_c(3\times(2w+1)\times W\times H\times M)$
Storage complexity	Unit: Number of floating-point numbers
Array $\mathbf{C}$	Maximum: $W\times H\times M$ Minimum: $(2w+1)\times W\times M$
Array $\mathbf{P}$	Maximum: $(2w+1)\times W\times M$ Minimum: $(2w+1)^2\times W$
Matrices $\mathbf{F}_l$ and $\mathbf{F}_r$	$L\times W$ entries, depending on the type of feature vector
Matrices $\mathbf{\Gamma}$ and $\mathbf{\Phi}$	W^2
Vector $\mathbf{a}$	M
Vector $\mathbf{w}$	$2w+1$

Algorithm 7 The proposed feature-vector-based cost aggregation algorithm

Inputs and parameters:
- rectified stereo image pair $\{I_l, I_r\}$ of $W \times H$ pixels in size
- disparity search range $\mathcal{D} = [d_{\min}, d_{\max}]$ of length M
- feature vector length L
- supporting window size $(2w+1) \times (2w+1)$

Outputs:
- disparity map $\mathbf{\Omega}$ of size $W \times H$

▷ Initialization of the following data to zero
- 3D cost volume array $\mathbf{C}$ of size $W \times H \times M$
- 3D per-pixel-weight array $\mathbf{P}$ of size $W \times H \times (2w+1)$
- Per-column-cost matrix $\mathbf{\Gamma}$ of size $W \times W$
- Per-column-weight matrix $\mathbf{\Phi}$ of size $W \times W$
- two matrices $\mathbf{F}_l$ and $\mathbf{F}_r$ of size $L \times W$
- weighted aggregated costs vector $\mathbf{a}$ of length M
- weights vector $\mathbf{w}$ of length $2w+1$

▷ Computation of $\mathbf{C}$ and $\mathbf{P}$
for each row y **do**
  Compute $\mathbf{F}_l^y$ and $\mathbf{F}_r^y$ using Eq. (7.6).
  Compute $C(x,y,d)$ for each x and $d \in \mathcal{D}$ using Eq. (7.5).
  Compute $P(x,y,z) = \exp\left(-\frac{|z|}{\sigma_1}\right)\exp\left[-\frac{1}{\sigma_2}c(\mathbf{f}(I_l;x,y),\mathbf{f}(I_l;x+z,y))\right]$ for each x and $z \in [-w,w]$.
end for

▷ Computation of the weighted aggregated costs
for each row $y \in [w+1, H-w]$ **do**
  if $y = w+1$ **then**
    Compute $\Phi_y(x+z,x) = \Phi_y(x,x+z) = \sum_{t=-w}^{w} P(x,y+t,z)$ for each x and $z \in [-w,w]$.
    Construct $\mathbf{\Gamma}_y$ using Eq. (7.8).
  else
    Update $\Phi_y(i,j) = \Phi_{y-1}(i,j) + P(j,y+w,i-j) - P(j,y-1-w,i-j)$.
    Update $\mathbf{\Gamma}_y$ using Eq. (7.9).
  end if
  for each x **do**
    Compute $w(z) = \Phi_y(x+z,x)$ for each $z \in [-w,w]$.
    Normalize $w(z)$ to sum to 1.
    Compute $a(d) = \sum_{z=-w}^{w} \frac{w(z)}{2w+1}\Gamma_y(x+z+d,x+z)$ for each $d \in \mathcal{D}$.
    Set $\Omega(x,y) = \arg\min_d a(d)$.
  end for
end for

7.3.2.3 Computation complexity

To discuss the algorithm complexity, three types of operations involved in the proposed method are distinguished from each other: (a) feature vector computation, (b) feature vector comparison, and (c) basic floating-point arithmetic operation. Operation (a) is the computation of a feature vector at a given pixel. Operation (b) is the computation of the dissimilarity between two feature vectors. Operation (c) obviously is much less computationally-intensive than Operations (a) and (b). For this reason, the symbols $O_a(1)$, $O_b(1)$ and $O_c(1)$ are used to denote the time complexity for each of these three operations, respectively.

The computation of the feature vectors for all pixels in I_l and I_r has a time complexity of $O_a(2\times W\times H)$. To construct the cost volume array $\mathbf{C}$, the term $c(\mathbf{f}(I_l;x,y),\mathbf{f}(I_r;x+d,y))$ needs to be computed for each location (x,y) in the left image and each disparity candidate d in $\mathcal{D}$, which leads to a time complexity of $O_b(W\times H\times M)$. Similarly, the construction of array $\mathbf{P}$ needs a time complexity of $O_b(W\times H\times[2w+1])$.

The initial construction of matrix $\mathbf{\Gamma}$ has a time complexity of $O_c(W\times[2w+1]^2\times M)$. Note that $W\times M$ is the number of valid entries in $\mathbf{\Gamma}$. Then, updating each valid entry requires only three single floating-point arithmetic operations (see Eq. (7.8)), which has a time complexity of $O_c(3\times W\times(H-2w-1)\times M)\approx O_c(3\times W\times H\times M)$ for this purpose. Similarly, the construction and update of matrix $\mathbf{\Phi}$ require a time complexity of $O_c(\frac{(2w+1)^2}{2}\times W)$ and $O_c(3/2\times W\times H\times(2w+1))$, respectively. Note that the $1/2$ factor is due to the symmetry of $\mathbf{\Phi}$.

Once $\mathbf{\Phi}$ and $\mathbf{\Gamma}$ are computed, for each pixel (x,y) in I_l, the evaluation of $\mathbf{a}$ needs $3\times(2w+1)\times M$ floating-point arithmetic operations. The evaluation of $\mathbf{w}$ requires an extra $(2w+2)$ floating-point arithmetic operations, which is negligible compared with that of evaluating $\mathbf{a}$. Therefore, for all the pixels in I_l, the time complexity for this step is $O_c(3\times(2w+1)\times W\times H\times M)$.

Now let us analyze the storage requirement for the proposed algorithm. The

cost volume array $\mathbf{C}$ and the array $\mathbf{P}$ require storage of $W \times H \times M$ and $W \times H \times (2w+1)$ floating-point numbers, respectively. They account for the largest share of storage requirement. However, if storage is limited, this requirement can be significantly reduced. In fact, only the top $(2w+1)$ rows participate in the initial construction of $\mathbf{\Gamma}$, and we only need a small portion of $\mathbf{C}$ corresponding to these $(2w+1)$ rows, which consists of $(2w+1) \times W \times M$ floating-point numbers. Afterwards, matrix $\mathbf{\Gamma}$ is iteratively updated by "removing" the oldest contributing "row" of C and adding a new one, which means only two "rows" of $\mathbf{C}$ need to be kept, or $2 \times W \times M$ floating-point numbers. Summarizing these two cases, it would be sufficient to use $(2w+1) \times W \times M$ floating-point numbers to dynamically keep those "rows" of $\mathbf{C}$ that are needed for the construction or update of $\mathbf{\Gamma}$. Similarly, $(2w+1)^2 \times W$ floating-point numbers are required for the construction or update of $\mathbf{P}$.

The matrices $\mathbf{F}_l$ and $\mathbf{F}_r$ both have $L \times W$ elements, and the storage requirement is dependent on the type of the chosen feature vector. For feature vectors such as DAISY and SIFT, each element is one floating-point number. For feature vectors such as BRIEF and BRISK, each element is one bit. The matrices $\mathbf{\Gamma}$ and $\mathbf{\Phi}$ each require W^2 floating-point numbers for storage, and can be re-used for each row. The storage requirement can be further reduced if their sparsity can be exploited. Finally, the vectors $\mathbf{a}$ and $\mathbf{w}$ require M and $(2w+1)$ floating-point numbers for storage, respectively. The time and storage complexity for the proposed method are summarized in Table 7.1.

7.4 Experimental results

This section begins with the introduction of two datasets in Section 7.4.1 and the DAISY descriptor in Section 7.4.2. Then, it proceeds to select the parameters of the proposed method in Section 7.4.3. Next, the proposed feature-vector-based weighting strategy is compared with several other weighting strategies in

Section 7.4.4. Finally, the performance of the proposed cost aggregation method is compared with the performance of two benchmark methods on the datasets in Section 7.4.5.

The proposed algorithm was implemented using C++. The experiments were done on a desktop computer equipped with an Intel Core i7-4770@3.40 GHz CPU, 8 GB memory, and 64-bit Windows 7 Enterprise operating system.

7.4.1 Wide-baseline stereo image data

The experiments in this work used two groups of wide baseline stereo image data: 1) the Fountain and HerzJesu Dataset, and 2) the 2014 Middlebury Stereo Dataset.

7.4.1.1 The Fountain and HerzJesu Dataset

The first group of test data is from the public dataset released by Strecha *et al*. [218]. Specifically, two datasets are used: the "Fountain" dataset and the "HerzJesu" dataset. Both datasets contain gray-scale images of 768×512 pixels, along with their ground-truth depth maps and occlusion maps. The camera calibration parameters associated with each gray-scale image are available, allowing these images to be rectified. The rectified images are also of 768×512 pixels.

For the "Fountain" dataset, eleven wide-baseline stereo images were tested in the experiments. One stereo image was used for the parameter selection experiment, presented in Section 7.4.3. The other ten stereo images were divided into two subsets, denoted as Fountain-A and Fountain-B. Each subset contains five consecutive images. For each subset, one image was considered as the left image while the other four images were considered as the right images. This was repeated five times, giving twenty stereo pairs for each subset. For the "HerzJesu" dataset, five images were selected to form another subset. In all, the first group of test data contains 60 stereo pairs, named "Fountain and HerzJesu Dataset" for brevity.

For this dataset, the performance of a given stereo matching method is measured using *precision*, *recall* and runtime. Let m be the number of *correctly* estimated non-occluded pixels, n the number of non-occluded pixels, and N the number of ground truth non-occluded pixels. For a given non-occluded pixel, if the relative error between its estimated depth value and the ground-truth depth value is less than 5%, the estimation is considered to be correct. The precision and recall are defined as

$$\begin{aligned} precision &= \frac{m}{n}, \\ recall &= \frac{n}{N}. \end{aligned} \tag{7.12}$$

7.4.1.2 The 2014 Middlebury Stereo Dataset

The second group of test data is from the 2014 Middlebury Stereo Dataset [219]. The stereo image pairs in this dataset were generated using a structured lighting system, and were meant to present new challenges for the next generation of stereo algorithms. Of the 33 stereo image pairs in the 2014 Middlebury Stereo Dataset, only 23 of them have accompanying ground-truth disparity maps. These 23 stereo pairs in their quarter resolution form the second group of test data, including: 1) adirondack, 2) jadeplant, 3) motorcycle, 4) piano, 5) pipes, 6) playroom, 7) playtable, 8) recycle, 9) shelves, 10) vintage, 11) backpack, 12) bicycle1, 13) cable, 14) classroom1, 15) couch, 16) flowers, 17) mask, 18) shopvac, 19) sticks, 20) storage, 21) sword1, 22) sword2, and 23) umbrella.

Because this dataset has no accompanying occlusion maps, the performance of a given method is measured by the *overall disparity estimation accuracy*, which is defined as the fraction of pixels with correctly estimated disparity values. If the difference between the estimated disparity and the ground-truth disparity of a pixel is less than two pixels, the disparity is considered as correctly estimated.

7.4.2 Implementation of the DAISY feature vector

In this work, the DAISY feature descriptor [190] is chosen to implement and test the proposed method. The DAISY descriptor gets its name from its flower-like shape. The center of the flower is located at the center of an image patch. There are Q concentric rings surrounding the flower center, each ring containing T evenly distributed circles. These $Q \times T$ circles form the flower petals. The interested reader is referred to Fig. 6 of [190] for a visual appearance of the DAISY descriptor. The flower center and petals are each described by a histogram of length H, which is the convolved orientation map computed at the flower center or a petal. Thus, a DAISY descriptor contains $H \times (Q \times T + 1)$ elements. The default parameters published in [190] were used in the experiments: $Q = 3$, $T = 8$, and $H = 8$, for a feature vector of 200 elements.

The DAISY feature descriptor has been shown to outperform SIFT and SURF in wide-baseline stereo matching [190]. In addition, DAISY is more computationally efficient than SIFT because it reuses the descriptor computation of other pixels. A disadvantage of the DAISY descriptor is that it is not scale- and rotation-invariant. However, since rectified images are used as input, the scale and rotation disparities between the stereo image pair are mostly compensated. The C++ implementation of the DAISY descriptor is publicly available[a].

For two DAISY descriptors $\mathbf{f}_1$ and $\mathbf{f}_2$, their dissimilarity is defined as

$$c(\mathbf{f}_1, \mathbf{f}_2) = \frac{1}{S} \sum_{k=1}^{S} \|\mathbf{f}_1^k - \mathbf{f}_2^k\|_2, \tag{7.13}$$

where S is the total number of non-occluded histograms, and $\mathbf{f}_1^k$ and $\mathbf{f}_2^k$ are the k-th normalized histogram of $\mathbf{f}_1$ and $\mathbf{f}_2$, respectively [190]. Of the $(Q \times T + 1) = 25$ histograms of a DAISY descriptor, some may be occluded because their corresponding petals lie outside the image plane. Hence, only the non-occluded histograms are used for matching. Each of the non-occluded histograms is normalized to

[a] http://cvlab.epfl.ch/software/daisy.

unity norm. The dissimilarity measure $c(\mathbf{f}_1, \mathbf{f}_2)$ ranges between 0 (perfect match) and 2 (complete non-match).

7.4.3 Parameter selection

The proposed method has two important parameters, the supporting window size (determined by w) and the threshold value τ. The algorithm was run with various combinations of w and τ on two stereo pairs shown in Figure 7.5. These two stereo pairs were not included in the Fountain and HerzJesu Dataset described in Section 7.4.1. The τ values were varied between 0.1 and 0.9 with a step of 0.1. The runtimes for different sizes of the supporting window were averaged. The results in terms of precision versus recall and average runtime are shown in Figure 7.6. Each τ value corresponds to a cross on the precision-recall curves. This figure shows that for every supporting window size, the recall increases and the precision decreases for increasing τ. Another observation is that a large supporting window size does not necessarily increase the performance in terms of precision-recall, despite an increase in runtime. As discussed in Section 7.3.2.1, a feature vector describes the statistics in an image patch around a pixel. For all the pixels inside a supporting window, the union of their corresponding image patches exceeds the supporting window itself. For this reason, one can use a small supporting window, instead of a large one as is often required by pixel-intensity-based stereo matching. Based on this analysis, a combination of $w = 3$ and $\tau = 0.3$ was used in the subsequent experiments.

Figure 7.5: Two stereo pairs used for parameter selection for the proposed method. (a) A "Fountain" stereo pair. (b) A "HerzJesu" stereo pair.

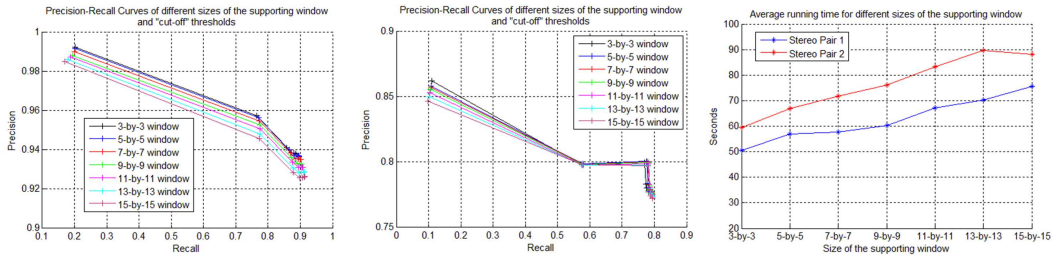


Figure 7.6: Performance of the proposed method with various combinations of w and τ on the two stereo pairs. Left: the precision-recall curves of different w and τ values for the first stereo pair. Middle: the precision-recall curves of different w and τ values for the second stereo pair. Right: average runtime for different values of w.

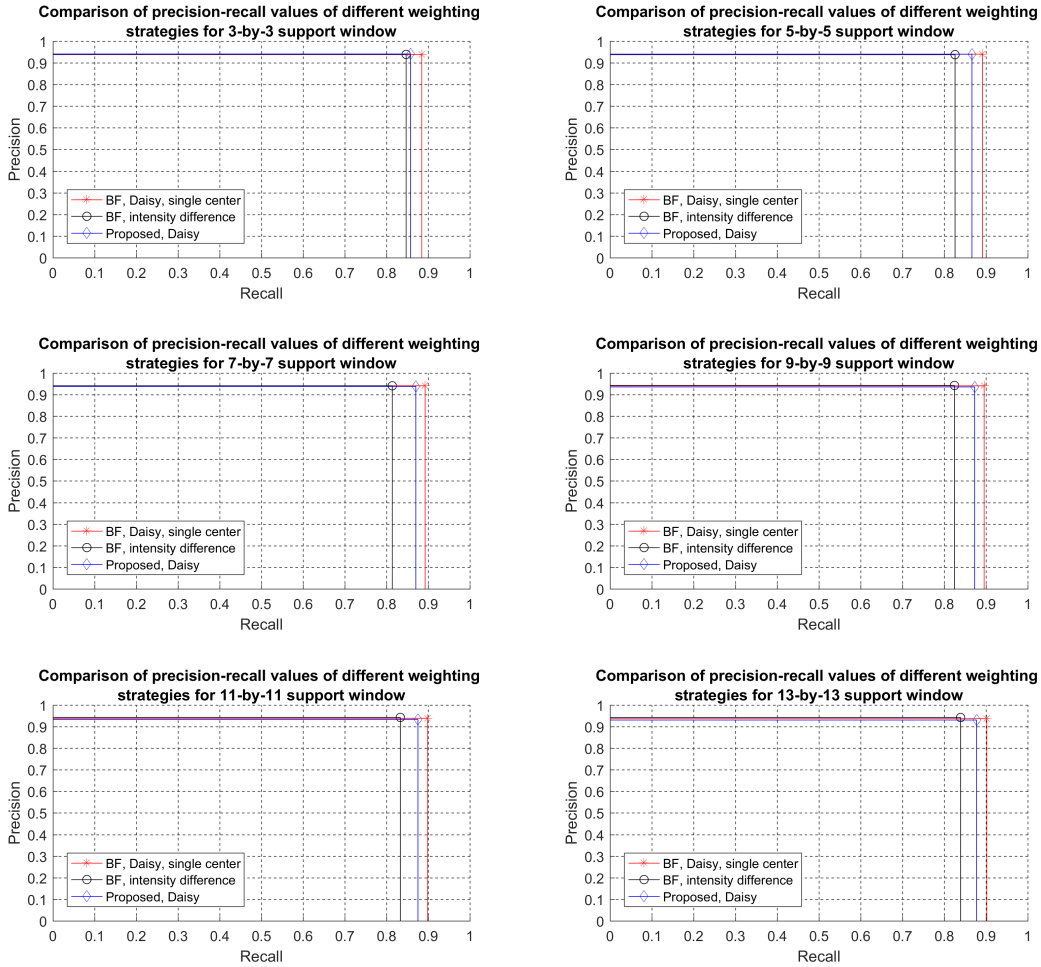


Figure 7.7: Performance comparison of the proposed and two other bilateral-filtering-based weighing strategies on the stereo pair in Figure 7.5 (a).

7.4.4 Analysis of weighting strategies

In this experiment, the proposed feature-vector-based weighting strategy is analyzed and compared with three other weighting strategies. The experiment used the two stereo pairs shown in Figure 7.5. Two weighting strategies are compared here: 1) the conventional bilateral filter with intensity difference [193], and 2) the bilateral filter with a single center. The first weighting strategy is represented by Eq. (7.3). The second weighting strategy is an extension of Eq. (7.3), by replacing the term $\|I_l(x,y) - I_l(x+i,y+j)\|_2$ with the feature-vector-based dissimilarity term $c(\mathbf{f}(I_l;x,y),\mathbf{f}(I_l;x+i,y+j))$. For a fair comparison, the DAISY feature vector was used to compute the initial cost volume in Eq. (7.1), instead of the truncated absolute differences. This way, the performance differences in cost aggregation were solely determined by the different weighting strategies. Figures 7.7 and 7.8 show the precision versus recall of the three weighting strategies on the two stereo pairs, for different supporting window sizes. The proposed weighting strategy significantly outperforms the bilateral filter with intensity difference. The proposed weighting strategy has slightly less number of recalls and processed much faster than the bilateral filter with single center.

Another weighting strategy to be compared here is based on the guided-image filter [196]. The cost volume was computed in two ways. One way was to use a combination of the truncated absolute difference of the color and the gradient, as in [196]. The other way was to use the DAISY feature vector. Either way, this weighting strategy failed to produce good disparity estimates on both stereo pairs, which suggests that the weighting strategy based on the guided-image filter may not be suitable for wide-baseline stereo matching.

7.4.5 Results on the two datasets and comparison with other methods

In this section, the proposed cost aggregation method is tested on the two datasets described in Section 7.4.1, and compared with two benchmark methods: Min *et al.*'s method [195], and the census transform [220, 221].

Min *et al*. developed an approximate strategy to optimize cost aggregation [195]. This strategy, originally based on the truncated absolute difference (TAD) matching cost, consists of two parts: "disparity candidate selection" and "joint histogram-based aggregation". In our implementation, the strategy was extended to feature vectors, by replacing the TAD matching costs with the dissimilarities between feature vectors. To enable a fair comparison, the DAISY descriptors were stored in the memory for the disparity candidate selection.

The census transform is one of the most popular techniques to compute matching costs for stereo vision. This method creates an encoded bit string for the pixels in a window. If the intensity of a pixel is lower than that of the center pixel of the window, the corresponding bit is set to one; otherwise, it is set to zero. In this way, the census transform describes the spatial structure in the window. A census-transformed image pair is matched by computing the Hamming distance between the bit strings. Our C++ implementation of the census transform was adapted from Banks and Corke's source code that accompanies their publication [222]. In the experiments, a census window size of 31×31 pixels was used, which is compatible with the computation of the DAISY feature vector.

Both benchmark methods rely on the left-right crosscheck to detect pixels with unreliable disparities. That is, the disparity map for both the left and right images are computed. First, for a pixel p in I_l, its counterpart q in I_r is found. Then, for pixel q, its counterpart p' in I_l is found. Finally, the disparity value for pixel p is considered as correctly estimated if $\|p - p'\| \leq \epsilon$, where ϵ is a small threshold. For a fair comparison, the left-right crosscheck also applied on the proposed method.

An evaluation of different values for ϵ indicated that $\epsilon = 2$ gave a good trade-off between the precision and recall rate.

7.4.5.1 Results on the Fountain and HerzJesu Dataset

The performance of the three methods is compared on the Fountain and herzJesu Dataset. The results are summarized in Table 7.2, including precision (denoted by "P"), recall (denoted by "R") and runtime obtained by the three methods. Min *et al.*'s method [195] yields the highest recall rate and the lowest precision rate among the three methods. The census transform provides the lowest recall rate and the highest precision rate among the three methods. The high precision rate by the census transform is attributed to its ability to capture the spatial structure of local windows using the encoded bit strings. The proposed method yields a middle rank in terms of precision and recall rates among the three methods. In terms of processing time, the proposed method is 2.42 times faster than Min *et al.*'s method, and 1.98 times faster than the census transform.

Table 7.2: Performance comparison of three methods on the Fountain and HerzJesu Dataset.

Method	Fountain-A		Fountain-B		HerzJesu		Time
	P	R	P	R	P	R	(sec)
Min *et al.* [195]	0.904	0.838	0.831	0.765	0.907	0.845	128
Census transform [222]	0.981	0.788	0.975	0.714	0.978	0.759	105
Proposed method	0.953	0.809	0.934	0.751	0.967	0.823	53

To further investigate their performance, the *proportion of correctly estimated non-occluded pixels* is also computed, which is the product of the precision and recall rates, see Eq. (7.12). The results, shown in Figure 7.9, indicate that the proposed method has a higher *precision* × *recall* score than the other two methods in most of the 60 stereo pairs. The average value of the *precision* × *recall* for the proposed method is 0.757, higher than that of the census transform (0.738) and Min *et al.*'s method (0.721).

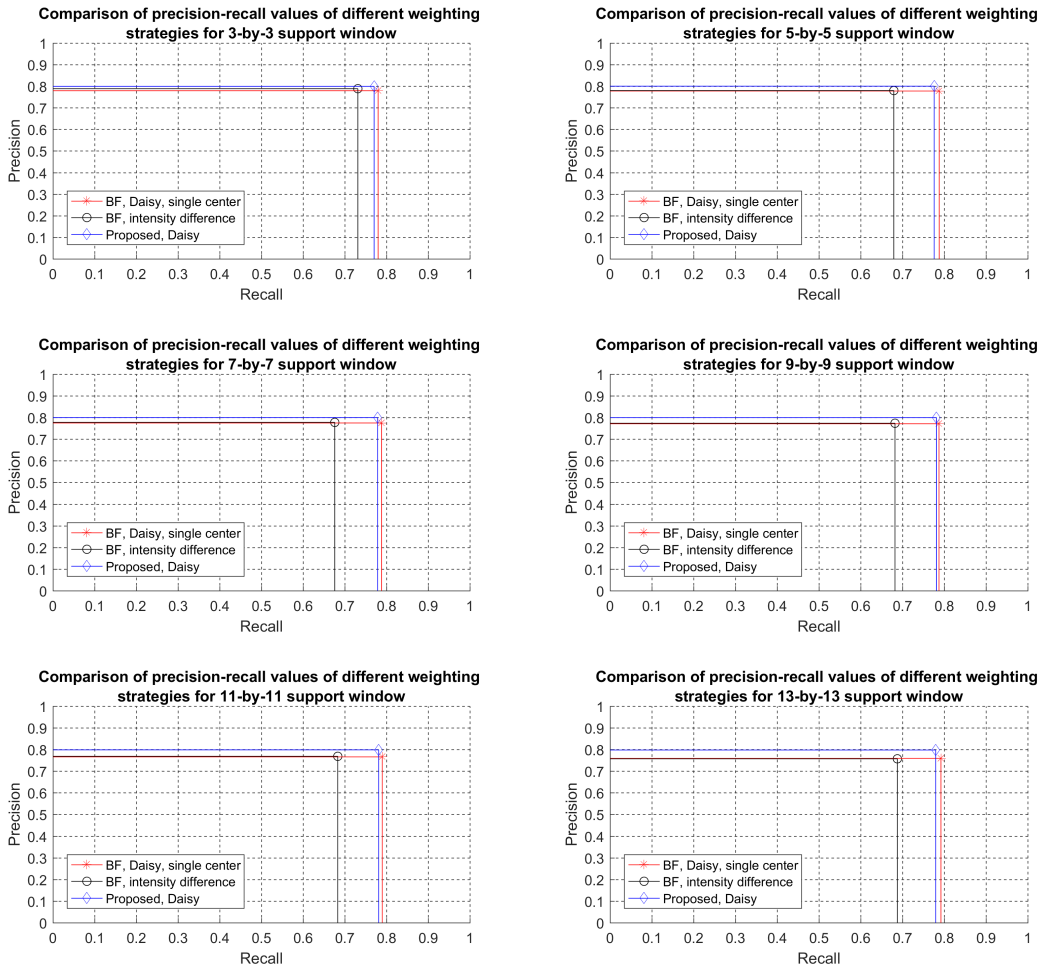


Figure 7.8: Performance comparison of the proposed and two other bilateral-filtering-based weighing strategies on the stereo pair in Figure 7.5 (b).

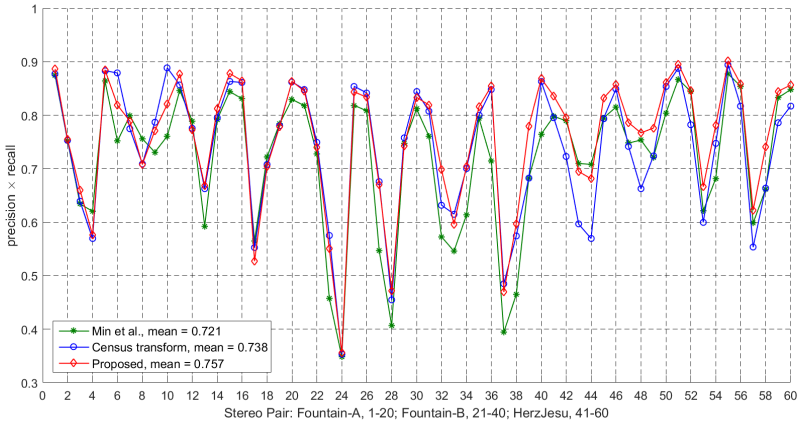


Figure 7.9: Performance comparison of three methods on the Fountain and HerzJesu Dataset.

Figure 7.10 presents representative results of depth estimation on this dataset. In this figure, column 1 and column 2 are the input stereo pair, column 3 is the output of the proposed method, where occluded pixels are shown in pink color. Column 4 compares the proposed method and Min *et al.*'s method [195]: If the depth of a pixel is correctly estimated by the proposed method but incorrectly estimated by Min *et al.*'s method, it is represented with *green* color; otherwise, it is represented with *red* color. Similarly, column 5 compares the proposed method with the census transform using the same color convention. In column 4 and column 5, a blue border around the image indicates the case where the proposed method performs worse than the other method. The distribution of the color pixels in column 4 and column 5 also reveal the stength of a given method in different parts of the image. Generally, the proposed method outperforms Min *et al.*'s method in most parts of the image. However, for sharp boundaries (*e.g.*, the boundaries between the red wall and the white wall in the last two rows of Figure 7.10), the census transform works better. This performance gap can be attributed to the different ways of forming the feature descriptors. The DAISY descriptor samples at sparse locations in a local area (only at the center and petals of the DAISY "flower"), while a bit string used in the census transform takes every location in the local area into account. Consequently, the DAISY descriptor may miss out some boundary pixels if its sample locations are far from them. This performance gap between the proposed method and the census transform can be reduced by using more advanced feature descriptors.

7.4.5.2 Results on the 2014 Middlebury Stereo Dataset

On the 2014 Middlebury Stereo Dataset, the disparity estimation accuracy obtained by the three methods is shown in Fig 7.11. For a fair comparison of the three methods, borders of half the census window width were discarded from the results. This is because the census transform generated zero pixels along the borders of census-transformed images. The proposed method and the census

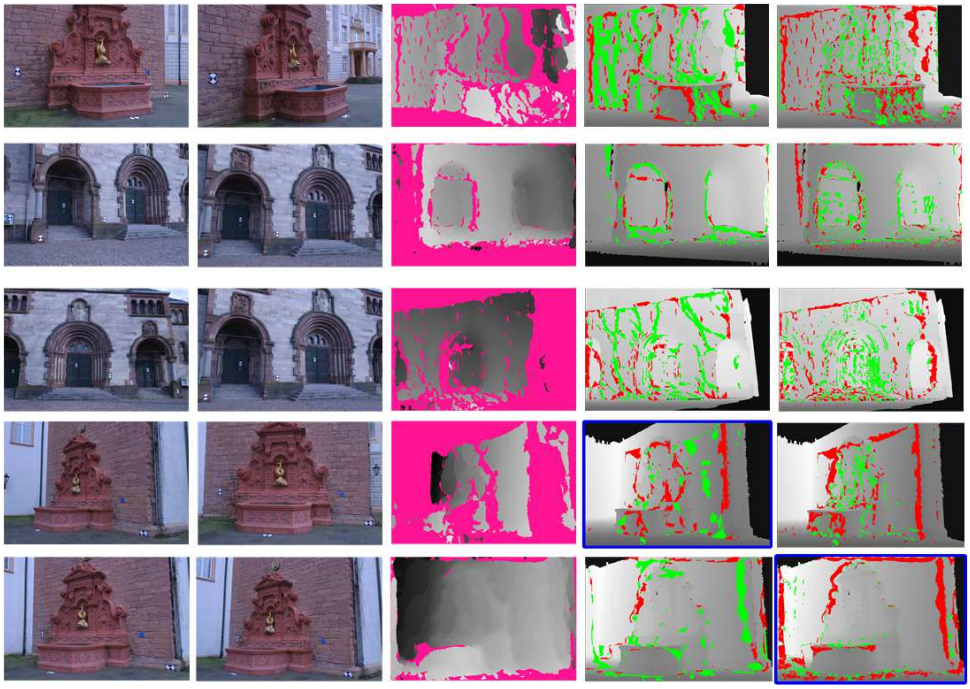


Figure 7.10: Representative examples of depth estimation from the Fountain and HerzJesu Dataset. Column 1 and column 2: the stereo pairs. Column 3: depth estimation results by the proposed method, where pink pixels denote occluded pixels. Column 4: comparison of the proposed method with Min *et al.*'s method [195]. If the depth of a pixel is correctly estimated by the proposed method but incorrectly by Min *et al.*'s method, the pixel is shown in *green* color; otherwise, it is shown in *red* color. Column 5: comparison of the proposed method with the census transform using the same color convention. A blue border around the image indicates the case where the proposed method performs worse compared to the other methods.

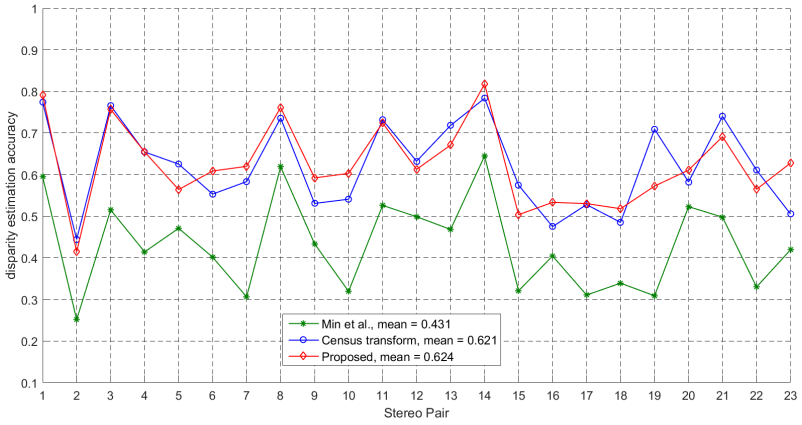


Figure 7.11: Performance comparison of three methods on the 2014 Middlebury Stereo Dataset.

transform perform significantly better than Min *et al.*'s method. The average disparity estimation accuracy for the proposed method, the census transform, and Min *et al.*'s method is 0.624, 0.621, and 0.433, respectively. Of the 23 stereo pairs, the proposed method achieves the highest disparity estimation accuracy for 13 stereo pairs. Note that the 2014 Middlebury Stereo Dataset contains less texture compared with the Fountain and HerzJesu Dataset. This result indicates that the proposed method and the census transform are more stable for textureless scenes.

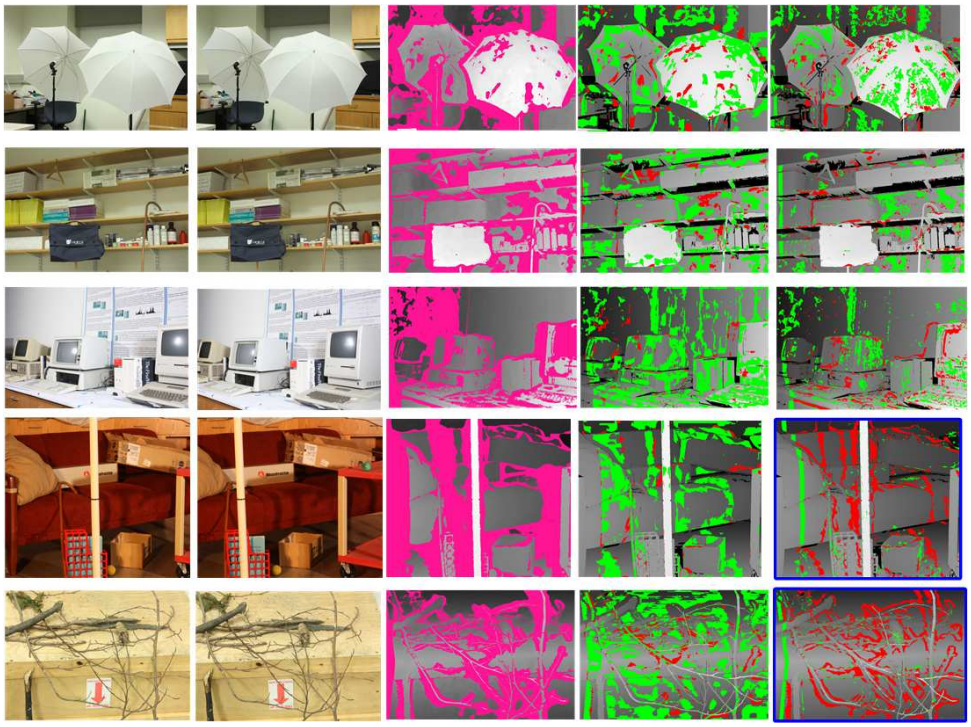


Figure 7.12: Representative examples of disparity estimation from the 2014 Middlebury Stereo Dataset. Column 1 and column 2: the stereo pairs. Column 3: disparity estimation results by the proposed method, where *pink* denote pixels with incorrect disparity estimations. Column 4: comparison of the proposed method with Min *et al.*'s method [195]. If the disparity of a pixel is correctly estimated by the proposed method but incorrectly by Min *et al.*'s method, the pixel is shown in *green* color; otherwise, it is shown in *red* color. Column 5: comparison of the proposed method with the census transform using the same color convention. A blue border around the image indicates the case where the proposed method performs worse compared to the other methods.

Figure 7.12 presents representative examples of disparity estimation on the 2014 Middlebury Stereo Dataset. Column 1 and column 2 are the input stereo pairs. Column 3 is the output of the proposed method, where pixels with incorrect disparity estimations are shown in pink. Column 4 and column 5 compare the

proposed method with Min *et al.*'s method and the census transform, using the same color convention as in Figure 7.12. However, because this dataset has ground-truth disparity maps instead of depth maps, the outputs are overlaid on the ground-truth disparity maps. The results again show that at sharp boundaries (*e.g.*, the "sticks" stereo pair in the last row of Figure 7.12), the proposed method may perform less successfully than the census transform.

7.5 Chapter summary

The preliminary step for 3D dynamic scene recognition is depth estimation. In this chapter, a feature-vector-based cost aggregation algorithm is proposed for wide-baseline stereo matching, and evaluated using the DAISY feature vector. The proposed algorithm improves the efficiency of cost aggregation by combining a per-column-cost matrix and a feature-vector-based weighting strategy. The chapter also presents a detailed analysis of both the time and storage complexity of the proposed method. The proposed method was extensively tested and compared with two benchmark methods on two wide-baseline datasets.

Chapter 8

Conclusion

Chapter contents

This thesis explores methods and techniques related with *dynamic scene recognition*. Most efforts of this research are devoted to video-based scene recognition. However, some efforts of this research are dedicated to stereo matching for depth estimation, which is a preliminary step towards 3D dynamic scene recognition. As the final part of the thesis, this chapter first summarizes the major research activities undertaken during this project (Section 8.1). Then, it gives concluding remarks and suggests directions for future work (Section 8.2).

8.1 Research summary

The research activities that have been documented in **Chapter 2** to **Chapter 7** of the thesis are summarized as follows.

- **Chapter 2**

  ① A comprehensive review of video-based dynamic scene recognition methods was conducted in this chapter. The methods discussed and

analyzed are roughly divided into two broad categories according to the features they use: handcrafted methods and deep-learning methods.

② Handcrafted methods traditionally dominated dynamic scene recognition, and provide insightful solution pipelines. However, deep-learning methods are current state-of-the-art in this field, and is thus the focus of this chapter.

③ Approaches and techniques meant for the more broad aim of video classification are also included in this chapter.

- **Chapter 3**

  ① A comprehensive review of mid-level representations was conducted in this chapter. The methods discussed and analyzed are roughly divided into three categories: 1) topic models, 2) part-based models and 3) bank-based representations.

  ② Part-based models, which seek those parts of an image that are more representative and discriminative than others, are the focus of this chapter.

  ③ Mid-level representations offer a useful alternative to global features for image or video representation.

- **Chapter 4**

  ① A part-based method was proposed to aggregate local features from video segments for dynamic scene recognition. The method consists of six steps: 1) feature initialization, 2) feature clustering, 3) part selection, 4) part refinement, 5) spatial and temporal information aggregation, and 6) classification. A pre-trained Fast R-CNN model is used to extract local convolutional features from the ROIs of training images. These features are clustered to locate representative parts. A set cover problem is then formulated to select the discriminative parts, which

are refined by fine-tuning the Fast R-CNN model. Local features from a video segment are extracted at different layers of the fine-tuned Fast R-CNN model, aggregated separately, and then concatenated into a global feature representation.

② Medium-sized auxiliary datasets containing static images were introduced to address the issue of appearance under-representation in the training videos. Steps 1-4 (feature initialization and clustering, part selection and refinement) of the proposed method are applied on this auxiliary dataset. The role of the auxiliary datasets in the performance of the proposed method was revealed in three experiments.

③ Extensive experiments show that the proposed method is very competitive with state-of-the-art methods that use global features extracted from the entire video frame or a video segment.

- **Chapter 5**

① A trajectory-based dynamic scene recognition method was proposed. This approach extracts densely and evenly distributed trajectories from a video segment. The fc6 features are extracted from the local regions around a trajectory using a pre-trained CNN model, forming a sequence of features. These sequences of fc6 features are used as input to an LSTM, called the local LSTM, to learn their temporal behavior. A trajectory is represented by the final time step of the output of the local LSTM. The local trajectory descriptor of the video segment is computed by aggregating the features of all the local trajectories in the video segment using the VLAD representation.

② The local LSTM is trained using synthetic trajectory data instead of real trajectory data, to avoid the impurity issue and the imbalance issue with the real training data. A systematic stop criterion is also proposed to control the quantity of synthetic trajectories. These techniques can

be extended to synthesize global trajectories, which are formed by fc6 features extracted from the whole video frames.

③ Beyond the classification problem, the "where" problem is explored to locate category-dependent discriminative trajectories in a video segment. Specifically, an optimization problem is formulated to learn all the discriminative part detectors of all the categories simultaneously.

- **Chapter 6**

① ACGANs use class labels along with latent samples as input to the generator. This property enables ACGANs to be used as a promising reconstruction-based classifier whereby an image can be classified according to the inferred class label.

② Three methods were designed to infer the input to the generator of an ACGAN, named i-ACGAN-r, i-ACGAN-d and i-ACGAN-e. The first two methods are "inverting" methods, which obtain the inverse mapping from an image to the class label and the latent sample by formulating a discrete-continuous optimization problem. i-ACGAN-r relaxes the discrete-continuous optimization problem by treating the class label as a continuous variable. By contrast, i-ACGAN-d solves for the discrete class label by decomposing an ACGAN into a series of regular GANs. Different from them, i-ACGAN-e directly infer both the class label and the latent sample by introducing an encoder into an ACGAN.

③ The methods were compared with two performance measures, the class recovery accuracy and the image reconstruction error. Experimental results of the three methods tested on three datasets, including a dynamic scene dataset, show that i-ACGAN-e outperforms the other two methods in terms of the class recovery accuracy criterion. However, the

images generated by i-ACGAN-r and i-ACGAN-d have smaller image reconstruction errors than i-ACGAN-e.

- **Chapter 7**

  ① In some applications such as autonomous driving, dynamic scene recognition may also be performed in a 3D scenario when the depth information of the scene is available.

  ② As a preliminary step towards designing a future 3D dynamic scene system, a feature-vector-based cost aggregation algorithm was developed for wide-baseline stereo matching. By combining a per-column-cost matrix and a feature-vector-based weighting strategy, the method enhances the cost aggregation efficiency.

8.2 Conclusion and future work

The main work undertaken in this research includes the following five aspects:

1. Conducted a comprehensive literature review of dynamic scene recognition and mid-level representations.
2. Proposed a part-based dynamic scene recognition method.
3. Proposed a trajectory-based dynamic scene recognition method.
4. Developed and compared three methods to infer the input to the generator of ACGANs.
5. Proposed a cost aggregation method for wide-baseline stereo matching.

The proposed methods were extensively tested and compared with existing methods. The results proved their effectiveness and efficiency. They have the potential to be used in real world applications. Despite these achievements, these methods can be improved further. Below is a list of future research directions:

- Disentangle camera movement from scene self-movement. Camera movement entwined with scene movement makes the recognition problem much more involving. Currently, it is difficult, if not impossible, to remove the camera movement from a natural dynamic scene. This is mainly because the optical-flow constraints commonly used in many visual tracking methods are usually violated in natural scenes such as fires, volcano eruptions and avalanches. Thus, investigating methods to better describe the motion of a natural scene is of great importance.

- Introduce domain adaptation to further improve the recognition performance. The benchmark video datasets used for dynamic scene recognition are quite limited in size. Most existing methods use transfer learning to address this issue. The rationale behind these methods is the ability of pre-trained CNN models, obtained by learning from millions of training data, can be transferred to represent and discriminate unseen data. The success of these approaches is built on the assumption that the training data is representative of the underlying data distribution. If this assumption is violated, these approaches may not work well. In case of this, domain adaptation can be considered, in which a model trained on a source distribution is used in a different (but related) target distribution.

- Explore discriminative trajectories to enhance classification performance. Inspired by part-based models, which seek more discriminative and representative parts in images, discriminative trajectories may further enhance the classification of dynamic scene videos. This can be realized by designing new feature aggregation methods to assign higher weights to more discriminative trajectories.

- Investigate the role of generative models in classification. When it comes to classification, discriminative models are usually preferred over generative models. One reason for the performance gap between these two types of

models can be attributed to the difficulty in modeling the true distribution of the data for a generative model. However, with the progress of deep learning, more advanced generative models have recently been proposed. GANs are state-of-the-art generative models. Their potential in classification is yet to be fully explored.

- Design a full 3D dynamic scene recognition system. 3D dynamic scene recognition is more complex than video-based dynamic scene recognition. Though a first-step attempt has been made in this research to develop a wide-baseline stereo matching approach, there is still a long way to go for the full development of a 3D dynamic scene recognition system. The next step down the path is reconstruction of the 3D scene using depth information.

Bibliography

[1] C. Yuan, Y. Zhang, and Z. Liu, "A survey on technologies for automatic forest fire monitoring, detection, and fighting using unmanned aerial vehicles and remote sensing techniques," *Canadian Journal of Forest Research*, vol. 45, no. 7, pp. 783–792, 2015.

[2] Y. Wang, W. L. Chao, D. Garg, B. Hariharan, M. Campbell, and K. Weinberger, "Pseudo-LiDAR from visual depth estimation: Bridging the gap in 3D object detection for autonomous driving," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019, pp. 8437–8445.

[3] M. Ravanbakhsh, M. Nabi, H. Mousavi, E. Sangineto, and N. Sebe, "Plug-and-play CNN for crowd motion analysis: An application in abnormal event detection," in *IEEE Winter Conference on Applications of Computer Vision (WACV)*, 2018, pp. 1689–1698.

[4] M. Ravanbakhsh, M. Nabi, E. Sangineto, L. Marcenaro, C. Regazzoni, and N. Sebe, "Abnormal event detection in videos using generative adversarial nets," in *IEEE International Conference on Image Processing (ICIP)*, 2017, pp. 1577–1581.

[5] M. Marszalek, I. Laptev, and C. Schmid, "Actions in context," in *IEEE*

Conference on Computer Vision and Pattern Recognition (CVPR), 2009, pp. 2929–2936.

[6] K. G. Derpanis and R. P. Wildes, "Dynamic texture recognition based on distributions of spacetime oriented structure," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2010, pp. 191–198.

[7] K. G. Derpanis, M. Lecce, K. Daniilidis, and R. P. Wildes, "Dynamic scene understanding: The role of orientation features in space and time in scene classification," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2012, pp. 1306–1313.

[8] K. G. Derpanis and R. P. Wildes, "Spacetime texture representation and recognition based on a spatiotemporal orientation analysis," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 34, no. 6, pp. 1193–1205, 2012.

[9] C. Feichtenhofer, A. Pinz, and R. P. Wildes, "Spacetime forests with complementary features for dynamic scene recognition," in *Proceedings of the British Machine Vision Conference (BMVC)*, 2013, pp. 1–12.

[10] ——, "Bags of spacetime energies for dynamic scene recognition," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2014, pp. 2681–2688.

[11] ——, "Dynamic scene recognition with complementary spatiotemporal features," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 38, no. 12, pp. 2389–2401, 2016.

[12] W. Freeman and F. Adelson, "The design and use of steerable filters," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 13, no. 9, pp. 891–906, 1991.

[13] Y. Rubner, C. Tomasi, and L. J. Guibas, "The Earth Mover's Distance as a metric for image retrieval," *International Journal of Computer Vision*, vol. 40, no. 2, pp. 99–121, 2000.

[14] J. Wang, J. Yang, K. Yu, F. Lv, T. Huang, and Y. Gong, "Locality-constrained linear coding for image classification," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2010, pp. 3360–3367.

[15] F. Perronnin, J. Sánchez, and T. Mensink, "Improving the Fisher kernel for large-scale image classification," in *European Conference on Computer Vision (ECCV)*, 2010, pp. 143–156.

[16] L. Breiman, "Random forests," *Machine Learning*, vol. 45, no. 1, pp. 5–32, 2001.

[17] Y. Quan, Y. Huang, and H. Ji, "Dynamic texture recognition via orthogonal tensor dictionary learning," in *IEEE International Conference on Computer Vision (ICCV)*, 2015, pp. 73–81.

[18] I. Laptev, "On space-time interest points," *International Journal of Computer Vision*, vol. 64, no. 2, pp. 107–123, 2005.

[19] C. Harris and M. Stephens, "A combined corner and edge detector," in *Proceedings of the Fourth Alvey Vision Conference*, 1998, pp. 147–151.

[20] N. Dalal and B. Triggs, "Histograms of oriented gradients for human detection," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2005, pp. 886–893.

[21] I. Laptev, M. Marszalek, C. Schmid, and B. Rozenfeld, "Learning realistic human actions from movies," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2008, pp. 1–8.

[22] D. G. Lowe, "Distinctive image features from scale-invariant keypoints," *International Journal of Computer Vision*, vol. 60, no. 2, pp. 91–110, 2004.

[23] G. Csurka, C. R. Dance, L. Fan, J. Willamowski, and C. Bray, "Visual categorization with bags of keypoints," in *European Conference on Computer Vision (ECCV) Workshop on Statistical Learning in Computer Vision*, 2004, pp. 1–22.

[24] A. B. Vasudevan, S. Muralidharan, S. P. Chintapalli, and S. Raman, "Dynamic scene classification using spatial and temporal cues," in *IEEE International Conference on Computer Vision (ICCV) Workshop*, 2013, pp. 803–810.

[25] N. Shroff, P. Turaga, and R. Chellappa, "Moving vistas: Exploiting motion for describing scenes," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2010, pp. 1911–1918.

[26] A. Oliva and A. Torralba, "Modeling the shape of the scene: A holistic representation of the spatial envelope," *International Journal of Computer Vision*, vol. 42, no. 3, pp. 145–175, 2001.

[27] G. Doretto, A. Chiuso, Y. N. Wu, and S. Soatto, "Dynamic textures," *International Journal of Computer Vision*, vol. 51, no. 2, pp. 91–109, 2003.

[28] A. Ravichandran, R. Chaudhry, and R. Vidal, "Categorizing dynamic textures using a bag of dynamical systems," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 35, no. 2, pp. 342–353, 2013.

[29] W. Huang, F. Sun, L. Cao, D. Zhao, H. Liu, and M. Harandi, "Sparse coding and dictionary learning with linear dynamical systems," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 3938–3947.

[30] K. D. Cock and B. D. Moor, "Subspace angles and distances between ARMA models," *System and Control Letters*, vol. 46, no. 4, pp. 265–270, 2002.

[31] C. Thériault, N. Thome, and M. Cord, "Dynamic scene classification: Learning motion descriptors with slow features analysis," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2013, pp. 2603–2610.

[32] L. Wiskott and T. J. Sejnowski, "Slow feature analysis: Unsupervised learning of invariances," *Neural Computation*, vol. 14, no. 4, pp. 715–770, 2002.

[33] P. Berkes and L. Wiskott, "Slow feature analysis yields a rich repertoire of complex cell properties," *Journal of Vision*, vol. 5, no. 6, pp. 579–602, 2005.

[34] Z. Zhang and D. Tao, "Slow feature analysis for human action recognition," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 34, no. 3, pp. 436–450, 2012.

[35] R. De Valois, E. Yund, and N. Hepler, "The orientation and direction selectivity of cells in macaque visual cortex," *Vision Research*, vol. 22, no. 5, pp. 531–544, 1982.

[36] P. Schäfer, "The BOSS is concerned with time series classification in the presence of noise," *Data Mining and Knowledge Discovery*, vol. 29, no. 6, p. 1505–1530, 2015.

[37] P. Schäfer and U. Leser, "Fast and accurate time series classification with WEASEL," in *ACM International Conference on Information and Knowledge Management*, 2017, pp. 637–646.

[38] ——, "Multivariate time series classification with WEASEL+MUSE," 2017. [Online]. Available: https://arxiv.org/abs/1711.11343

[39] X. Peng, L. Wang, X. Wang, and Y. Qiao, "Bag of visual words and fusion methods for action recognition: comprehensive study and good practice," *Computer Vision and Image Understanding*, vol. 150, no. 9, pp. 109–125, 2016.

[40] H. Wang, A. Kläser, C. Schmid, and C. L. Liu, "Dense trajectories and motion boundary descriptors for action recognition," *International Journal of Computer Vision*, vol. 103, no. 1, pp. 60–79, 2013.

[41] H. Wang and C. Schmid, "Action recognition with improved trajectories," in *IEEE International Conference on Computer Vision (ICCV)*, 2013, pp. 3551–3558.

[42] N. Dalal, B. Triggs, and C. Schmid, "Human detection using oriented histograms of flow and appearance," in *European Conference on Computer Vision (ECCV)*, 2006, pp. 428–441.

[43] M. R. Khokher, A. Bouzerdoum, and S. L. Phung, "A super descriptor tensor decomposition for dynamic scene recognition," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 29, no. 4, pp. 1063–1076, 2019.

[44] K. Papadopoulos, G. Demisse, E. Ghorbel, M. Antunes, D. Aouada, and B. Ottersten, "Localized trajectories for 2D and 3D action recognition," *Sensors*, vol. 19, p. 3503, 2019.

[45] M. Harandi, M. Salzmann, and M. Baktashmotlagh, "Beyond Gauss: Image-set matching on the Riemannian manifold of PDFs," in *IEEE International Conference on Computer Vision (ICCV)*, 2015, pp. 4112–4120.

[46] M. Harandi, R. Hartley, C. Shen, B. Lovell, and C. Sanderson, "Extrinsic methods for coding and dictionary learning on Grassmann manifolds," *International Journal of Computer Vision*, vol. 114, no. 2-3, pp. 113–136, 2015.

[47] M. Faraki, M. Harandi, and F. Porikli, "More about VLAD: A leap from Euclidean to Riemannian manifolds," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015, pp. 4951–4960.

[48] H. Jégou, F. Perronnin, M. Douze, J. Sánchez, P. Pérez, and C. Schmid, "Aggregating local image descriptors into compact codes," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 34, no. 9, pp. 1704–1716, 2012.

[49] X. Qi, C. Li, G. Zhao, X. Hong, and M. Pietikäinen, "Dynamic texture and scene classification by transferring deep image features," *Neurocomputing*, vol. 171, pp. 1230–1241, 2016.

[50] S. Hong, J. Ryu, W. Im, and H. S. Yang, "D3: Recognizing dynamic scenes

with deep dual descriptor based on key frames and key segments," *Neurocomputing*, vol. 273, pp. 611–621, 2018.

[51] Y. Huang, X. Cao, Q. Wang, B. Zhang, X. Zhen, and X. Li, "Long-short term features for dynamic scene classification," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 29, no. 4, pp. 1038–1047, 2019.

[52] C. Feichtenhofer, A. Pinz, and R. P. Wildes, "Temporal residual networks for dynamic scene recognition," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017, pp. 7435–7444.

[53] A. Gangopadhyay, S. M. Tripathi, I. Jindal, and S. Raman, "Dynamic scene classification using convolutional neural networks," in *IEEE Global Conference on Signal and Information Processing*, 2016, pp. 1255–1259.

[54] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," in *Conference and Workshop on Neural Information Processing Systems (NIPS)*, 2012, pp. 1097–1105.

[55] ——, "ImageNet classification with deep convolutional neural networks," *Communications of the ACM*, vol. 60, no. 6, pp. 84–90, 2017.

[56] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," in *International Conference on Learning Representations (ICLR)*, 2015.

[57] J. Deng, W. Dong, R. Socher, L. Li, K. Li, and L. Fei-Fei, "ImageNet: A large-scale hierarchical image database," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2009, pp. 248–255.

[58] S. Ji, W. Xu, M. Yang, and K. Yu, "3D convolutional neural networks for human action recognition," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 35, no. 1, pp. 221–231, 2013.

[59] A. Karpathy, G. Toderici, S. Shetty, T. Leung, R. Sukthankar, and L. Fei-Fei, "Large-scale video classification with convolutional neural networks," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2014, pp. 1725–1732.

[60] D. Tran, L. Bourdev, R. Fergus, L. Torresani, and M. Paluri, "Learning spatiotemporal features with 3D convolutional networks," in *IEEE International Conference on Computer Vision (ICCV)*, 2015, pp. 4489–4497.

[61] D. Tran, H. Wang, L. Torresani, J. Ray, Y. LeCun, and M. Paluri, "A closer look at spatiotemporal convolutions for action recognition," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018, pp. 6450–6459.

[62] J. Carreira and A. Zisserman, "Quo vadis, action recognition? A new model and the Kinetics dataset," 2018. [Online]. Available: https://arxiv.org/abs/1705.07750

[63] J. Arunnehru, G. Chamundeeswari, and S. P. Bharathi, "Human action recognition using 3D convolutional neural networks with 3D motion cuboids in surveillance videos," *Procedia Computer Science*, vol. 133, pp. 471–477, 2018.

[64] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 770–778.

[65] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015, pp. 1–9.

[66] K. Simonyan and A. Zisserman, "Two-stream convolutional networks for

action recognition in videos," in *Conference and Workshop on Neural Information Processing Systems (NIPS)*, 2014, pp. 568–576.

[67] G. W. Taylor, R. Fergus, Y. LeCun, and C. Bregler, "Convolutional learning of spatio-temporal features," in *European Conference on Computer Vision (ECCV)*, 2010, pp. 140–153.

[68] Q. V. Le, W. Y. Zou, S. Y. Yeung, and A. Y. Ng, "Learning hierarchical invariant spatio-temporal features for action recognition with independent subspace analysis," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2011, pp. 3361–3368.

[69] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[70] J. Donahue, L. A. Hendricks, M. Rohrbach, S. Venugopalan, S. Guadarrama, K. Saenko, and T. Darrell, "Long-term recurrent convolutional networks for visual recognition and description," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015, pp. 2625–2634.

[71] J. Y. H. Ng, M. Hausknecht, S. Vijayanarasimhan, O. Vinyals, R. Monga, and G. Toderici, "Beyond short snippets deep networks for video classification," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015, pp. 4694–4702.

[72] Y. Yu, X. Si, C. Hu, and J. Zhang, "A review of recurrent neural networks: LSTM cells and network architectures," *Neural computation*, vol. 31, no. 7, pp. 1235–1270, 2019.

[73] F. Karim, S. Majumdar, H. Darabi, and S. Chen, "LSTM fully convolutional networks for time series classification," *IEEE Access*, vol. 6, pp. 1662–1669, 2018.

[74] K. Cho, B. van Merrienboer, C. Gulcehre, F. Bougares, H. Schwenk, and Y. Bengio, "Learning phrase representations using RNN encoder-decoder for statistical machine translation," in *Empirical Methods in Natural Language Processing (EMNLP)*, 2014.

[75] M. Schuster and K. K. Paliwal, "Bidirectional recurrent neural networks," *IEEE Transactions on Signal Processing*, vol. 45, no. 11, pp. 2673–2681, 1997.

[76] Z. C. Lipton, J. Berkowitz, and C. Elkan, "A critical review of recurrent neural networks for sequence learning," 2015. [Online]. Available: https://arxiv.org/abs/1506.00019

[77] M. MacKay, P. Vicol, J. Ba, and R. Grosse, "Reversible recurrent neural networks," in *Conference and Workshop on Neural Information Processing Systems (NIPS)*, 2018, pp. 9043–9054.

[78] C. Feichtenhofer, A. Pinz, and A. Zisserman, "Convolutional two-stream network fusion for video action recognition," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 1933–1941.

[79] L. Wang, Y. Xiong, Z. Wang, Y. Qiao, D. Lin, X. Tang, and L. Van Gool, "Temporal segment networks: towards good practices for deep action recognition," in *European Conference on Computer Vision (ECCV)*, 2016, pp. 20–36.

[80] Z. Lan, Y. Zhu, A. G. Hauptmann, and S. Newsam, "Deep local video feature for action recognition," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017, pp. 1–7.

[81] M. D. Zeiler and R. Fergus, "Visualizing and understanding convolutional networks," in *European Conference on Computer Vision (ECCV)*, 2014, pp. 818–833.

[82] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network

training by reducing internal covariate shift," in *International Conference on International Conference on Machine Learning (ICML)*, 2015, pp. 448–456.

[83] Z. Li, K. Gavrilyuk, E. Gavves, M. Jain, and C. G. M. Snoek, "VideoLSTM convolves, attends and flows for action recognition," *Computer Vision and Image Understanding*, vol. 166, no. 3, pp. 41–50, 2018.

[84] P. Wang, Y. Cao, C. Shen, L. Liu, and H. T. Shen, "Temporal pyramid pooling-based convolutional neural network for action recognition," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 27, no. 12, pp. 2613–2622, 2017.

[85] L. Liu, P. Wang, C. Shen, L. Wang, A. van den Hengel, C. Wang, and H. Shen, "Compositional model based fisher vector coding for image classification," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 39, no. 12, pp. 2335–2348, 2017.

[86] Z. Tu, W. Xie, Q. Qin, R. Poppe, R. C. Veltkamp, B. Li, and J. Yuan, "Multi-stream CNN: Learning representations based on human-related regions for action recognition," *Pattern Recognition*, vol. 79, no. 7, pp. 32–43, 2018.

[87] L. Fei-Fei and P. Perona, "A Bayesian hierarchical model for learning natural scene categories," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2005, pp. 524–531.

[88] D. Blei and M. Jordan, "Latent Dirichlet Allocation," *Journal of Machine Learning Research*, vol. 3, pp. 993–1022, 2003.

[89] J. Sivic, B. C. Russell, A. A. Efros, A. Zisserman, and W. T. Freeman, "Discovering object categories in image collections," in *IEEE International Conference on Computer Vision (ICCV)*, 2005, pp. 370–377.

[90] J. Zeng, W. K. Cheung, and J. Liu, "Learning topic models by belief propaga-

tion," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 35, no. 5, pp. 1121–1134, 2013.

[91] C. M. Bishop, *Pattern Recognition and Machine Learning*. New York: Springer, 2006.

[92] C. Liu, H. Lin, S. Gong, Y. ji, and Q. Liu, "Learning topic of dynamic scene using belief propagation and weighted visual words approach," *Soft Computing*, vol. 19, no. 1, pp. 71–84, 2015.

[93] N. Rasiwasia and N. Vasconcelos, "Latent Dirichlet Allocation models for image classification," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 35, no. 11, pp. 2665–2679, 2013.

[94] F. Rodrigues, M. Lourenço, B. Ribeiro, and F. C. Pereira, "Learning supervised topic models for classification and regression from crowds," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 39, no. 12, pp. 2409–2422, 2017.

[95] Z. Pan, Y. Liu, G. Liu, M. Guo, and Y. Li, "Topic Network: Topic model with deep learning for image classification," in *International Conference on Knowledge Science, Engineering and Management*, 2015, pp. 525–534.

[96] A. G. Sorkhi, H. Hassanpour, and M. Fateh, "A comprehensive system for image scene classification," *Multimedia Tools and Applications*, vol. 2020, pp. https://doi.org/10.1007/s11 042–019–08 264–y, 2020.

[97] T. Hofmann, "Unsupervised learning by probabilistic latent semantic analysis," *Machine Learning*, vol. 42, no. 1-2, pp. 177–196, 2001.

[98] A. Bosch, A. Zisserman, and X. Munoz, "Scene classification using a hybrid generative/discriminative approach," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 30, no. 4, pp. 712–727, 2008.

[99] P. F. Felzenszwalb, R. B. Girshick, D. McAllester, and D. Ramanan, "Object detection with discriminatively trained part-based models," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 32, no. 9, pp. 1627–1645, 2010.

[100] R. Girshick, F. Iandola, T. Darrell, and J. Malik, "Deformable part models are convolutional neural networks," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015, pp. 437–446.

[101] S. Singh, A. Gupta, and A. A. Efros, "Unsupervised discovery of mid-level discriminative patches," in *European Conference on Computer Vision (ECCV)*, 2012, pp. 73–86.

[102] J. Sun and J. Ponce, "Learning dictionary of discriminative part detectors for image categorization and cosegmentation," *International Journal of Computer Vision*, vol. 120, no. 2, pp. 111–133, 2016.

[103] U. von Luxburg, "A tutorial on spectral clustering," *Statistics and Computing*, vol. 17, no. 4, pp. 395–416, 2007.

[104] W. W. Chen, Y. Song, H. Bai, C. J. Lin, and E. Y. Chang, "Parallel spectral clustering in distributed systems," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 33, no. 3, pp. 568–586, 2011.

[105] C. Doersch, S. Singh, A. Gupta, J. Sivic, and A. A. Efros, "What makes Paris look like Paris?" *ACM Transactions on Graphics*, vol. 31, no. 4, p. Article No. 101, 2012.

[106] L. Bourdev and J. Malik, "Poselets: Body part detectors trained using 3D human pose annotations," in *IEEE International Conference on Computer Vision (ICCV)*, 2009, pp. 1365–1372.

[107] M. Juneja, A. Vedaldi, C. V. Jawahar, and A. Zisserman, "Blocks that shout:

distinctive parts for scene classification," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2013, pp. 923–930.

[108] B. Hariharan, J. Malik, and D. Ramanan, "Discriminative decorrelation for clustering and classification," in *European Conference on Computer Vision (ECCV)*, 2012, pp. 459–472.

[109] T. Malisiewicz, A. Gupta, and A. A. Efros, "Ensemble of exemplar-SVMs for object detection and beyond," in *IEEE International Conference on Computer Vision (ICCV)*, 2011, pp. 89–96.

[110] A. Jain, A. Gupta, M. Rodriguez, and L. S. Davis, "Representing videos using mid-level discriminative patches," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2013, pp. 2571–2578.

[111] I. Endres, K. J. Shih, J. Jiaa, and D. Hoiem, "Learning collections of part models for object recognition," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2013, pp. 939–946.

[112] C. N. J. Yu and T. Joachims, "Learning structural SVMs with latent variables," in *International Conference on International Conference on Machine Learning (ICML)*, 2009, pp. 1169–1176.

[113] M. Pandey and S. Lazebnik, "Scene recognition and weakly supervised object localization with deformable part-based models," in *IEEE International Conference on Computer Vision (ICCV)*, 2011, pp. 1307–1314.

[114] A. Beck and M. Teboulle, "A fast iterative shrinkage-thresholding algorithm for linear inverse problems," *SIAM Journal on Imaging Sciences*, vol. 2, no. 1, pp. 183–202, 2009.

[115] S. N. Parizi, J. G. Oberlin, and P. F. Felzenszwalb, "Reconfigurable models for scene recognition," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2012, pp. 2775–2782.

[116] J. Sivic and A. Zisserman, "Efficient visual search of videos cast as text retrieval," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 31, no. 4, pp. 591–606, 2009.

[117] Y. Li, L. Liu, C. Shen, and A. van den Hengel, "Mid-level deep pattern mining," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015, pp. 971–980.

[118] ——, "Mining mid-level visual patterns with deep CNN activations," *International Journal of Computer Vision*, vol. 121, no. 3, pp. 344–364, 2017.

[119] C. Zhang and S. Zhang, *Association Rule Mining Models and Algorithms*. New York: Springer, 2002.

[120] R. Agrawal and R. Srikant, "Fast algorithms for mining association rules in large databases," in *Proceedings of the 20th International Conference on Very Large Data Bases (VLDB)*, 1994, pp. 487–499.

[121] C. Doersch, A. Gupta, and A. A. Efros, "Mid-level visual element discovery as discriminative mode seeking," in *Conference and Workshop on Neural Information Processing Systems (NIPS)*, 2013, pp. 494–502.

[122] R. Sicre and F. Jurie, "Discriminative part model for visual recognition," *Computer Vision and Image Understanding*, vol. 141, no. 12, pp. 28–37, 2015.

[123] R. Sicre, Y. Avrithis, E. Kijak, and F. Jurie, "Unsupervised part learning for visual recognition," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017, pp. 6271–6279.

[124] H. Lobel, R. Vidal, and A. Soto, "Hierarchical joint max-margin learning of mid and top level representations for visual recognition," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2013, pp. 1697–1704.

[125] A. L. Yuille and A. Rangarajan, "The concave-convex procedure," *Neural Computation*, vol. 15, no. 4, pp. 915–936, 2003.

[126] S. N. Parizi, A. Vedaldi, A. Zisserman, and P. Felzenszwalb, "Automatic discovery and optimization of parts for image classification," in *International Conference on Learning Representations (ICLR)*, 2015.

[127] Z. Zuo, G. Wang, B. Shuai, L. Zhao, Q. Yang, and X. Jiang, "Learning discriminative and shareable features for scene classification," in *European Conference on Computer Vision (ECCV)*, 2014, pp. 552–568.

[128] P. Kulkarni, F. Jurie, J. Zepeda, P. Pérez, and L. Chevallier, "SPLeaP: soft pooling of learned parts for image classification," in *European Conference on Computer Vision (ECCV)*, 2016, pp. 329–345.

[129] L. J. Li, H. Su, Y. Lim, and L. Fei-Fei, "Object Bank: An object-level image representation for high-level visual recognition," *International Journal of Computer Vision*, vol. 107, no. 1, pp. 20–39, 2014.

[130] S. Sadanand and J. J. Corso, "Action bank: A high-level representation of activity in video," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2012, pp. 1234–1241.

[131] Z. Wang, L. Wang, Y. Wang, B. Zhang, and Y. Qiao, "Weakly supervised PatchNets: Describing and aggregating local patches for scene recognition," *IEEE Transactions on Image Processing*, vol. 26, no. 4, pp. 2028–2041, 2017.

[132] X. Cheng, J. Lu, J. Feng, B. Yuan, and J. Zhou, "Scene recognition with objectness," *Pattern Recognition*, vol. 74, no. 2, pp. 474–487, 2018.

[133] H. Zhu, "Massive-scale image retrieval based on deep visual feature representation," *Journal of Visual Communication and Image Representation*, vol. 70, no. 7, p. 102738, 2020.

[134] S. Bai and H. Tang, "Categorizing scenes by exploring scene part information without constructing explicit models," *Neurocomputing*, vol. 281, pp. 160–168, 2018.

[135] S. Lazebnik, C. Schmid, and J. Ponce, "Beyond bags of features: spatial pyramid matching for recognizing natural scene categories," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2006, pp. 2169–2178.

[136] A. Quattoni and A. Torralba, "Recognizing indoor scenes," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2009, pp. 413–420.

[137] A. S. Razavian, H. Azizpour, J. Sullivan, and S. Carlsson, "CNN features off-the-shelf: An astounding baseline for recognition," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2014, pp. 512–519.

[138] X. Wei, S. L. Phung, and A. Bouzerdoum, "Visual descriptors for scene categorization: experimental evaluation," *Artificial Intelligence Review*, vol. 45, no. 3, pp. 753–764, 2016.

[139] L. Du and H. Ling, "Dynamic scene classification using redundant spatial scenelets," *IEEE Transactions on Cybernetics*, vol. 46, no. 9, pp. 2156–2165, 2016.

[140] R. Girshick, "Fast R-CNN," in *IEEE International Conference on Computer Vision (ICCV)*, 2015, pp. 1440–1448.

[141] J. R. R. Uijlings, K. E. A. van de Sande, T. Gevers, and A. W. M. Smeulders, "Selective search for object recognition," *International Journal of Computer Vision*, vol. 104, no. 2, pp. 154–171, 2013.

[142] D. P. Williamson, "Lecture notes on approximation algorithms," *Technical Report*, 1998.

[143] R. Fan, K. Chang, C. Hsieh, X. Wang, and C. Lin, "LIBLINEAR: A library for large linear classification," *The Journal of Machine Learning Research*, vol. 9, no. 2, pp. 1871–1874, 2008.

[144] B. Zhou, A. Lapedriza, J. Xiao, A. Torralba, and A. Oliva, "Learning deep features for scene recognition using places database," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 40, no. 6, pp. 1452–1464, 2018.

[145] M. Friedman, "A comparison of alternative tests of significance for the problem of *m* rankings," *The Annals of Mathematical Statistics*, vol. 11, no. 1, pp. 86–92, 1940.

[146] C. Liu, J. Yuen, and A. Torralba, "SIFT flow: dense correspondence across scenes and its applications," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 33, no. 5, pp. 978–994, 2011.

[147] O. Vinyals, A. Toshev, S. Bengio, and D. Erhan, "Show and tell: Lessons learned from the 2015 MSCOCO image captioning challenge," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 39, no. 4, pp. 652–663, 2017.

[148] H. Kuehne, H. J. Huang, E. Garrote, T. Poggio, and T. Serre, "HMDB: A large video database for human motion recognition," in *IEEE International Conference on Computer Vision (ICCV)*, 2011, pp. 2556–2563.

[149] G. A. Sigurdsson, G. Varol, X. Wang, A. Farhadi, I. Laptev, and A. Gupta, "Hollywood in homes: Crowdsourcing data collection for activity understanding," in *European Conference on Computer Vision (ECCV)*, 2016, pp. 510–526.

[150] A. Creswell, T. White, and V. Dumoulin, "Generative adversarial networks: An overview," *IEEE Signal Processing Magazine*, vol. 35, no. 1, pp. 53–65, 2018.

[151] C. M. Bishop and J. Lasserre, "Generative or discriminative? Getting the best of both worlds," *Bayesian Statistics*, vol. 8, pp. 3–24, 2007.

[152] A. Y. Ng and M. I. Jordan, "On discriminative vs. generative classifiers: A comparison of logistic regression and naive Bayes," in *Conference and Workshop on Neural Information Processing Systems (NIPS)*, 2001, pp. 841–848.

[153] D. P. Kingma and M. Welling, "Auto-encoding variational Bayes," in *International Conference on Learning Representations (ICLR)*, 2014.

[154] A. Makhzani, J. Shlens, N. Jaitly, and I. Goodfellow, "Adversarial autoencoders," in *International Conference on Learning Representations (ICLR)*, 2015.

[155] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," in *Conference and Workshop on Neural Information Processing Systems (NIPS)*, 2014, pp. 2672–2680.

[156] A. Odena, C. Olah, and J. Shlens, "Conditional image synthesis with auxiliary classifier GANs," in *International Conference on International Conference on Machine Learning (ICML)*, 2017, pp. 2642–2651.

[157] Z. C. Lipton and S. Tripathi, "Precise recovery of latent vectors from generative adversarial networks," in *International Conference on Learning Representations (ICLR)*, 2017.

[158] A. Creswell and A. A. Bharath, "Inverting the generator of a generative adversarial network," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 30, no. 7, pp. 1967–1974, 2019.

[159] J. Donahue, P. Krähenbühl, and T. Darrell, "Adversarial feature learning," in *International Conference on Learning Representations (ICLR)*, 2017.

[160] V. Dumoulin, I. Belghazi, B. Poole, O. Mastropietro, A. Lamb, M. Arjovsky, and A. Courville, "Adversarially learned inference," in *International Conference on Learning Representations (ICLR)*, 2017.

[161] P. Belotti, C. Kirches, S. Leyffer, J. Linderoth, J. Luedtke, and A. Mahajan, "Mixed-integer nonlinear optimization," *Acta Numerica*, vol. 29, pp. 1–131, 2013.

[162] J. Hu, L. Shen, and G. Sun, "Squeeze-and-excitation networks," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018, pp. 7132–7141.

[163] W. Rawat and Z. Wang, "Deep convolutional neural networks for image classification: A comprehensive review," *Neural Computation*, vol. 29, no. 9, pp. 2352–2449, 2017.

[164] D. P. Kingma, D. J. Rezende, S. Mohamed, and M. Welling, "Semi-supervised learning with deep generative models," in *Conference and Workshop on Neural Information Processing Systems (NIPS)*, 2014, pp. 3581–3589.

[165] M. Mirza and S. Osindero, "Conditional generative adversarial nets," 2014. [Online]. Available: https://arxiv.org/abs/1411.1784

[166] X. Chen, Y. Duan, R. Houthooft, J. Schulman, I. Sutskever, and P. Abbeel, "InfoGAN: interpretable representation learning by information maximizing generative adversarial nets," in *Conference and Workshop on Neural Information Processing Systems (NIPS)*, 2016, pp. 2180–2188.

[167] M.-Y. Liu and O. Tuzel, "Coupled generative adversarial networks," in *Conference and Workshop on Neural Information Processing Systems (NIPS)*, 2016, pp. 469–477.

[168] T. Miyato and M. Koyama, "cGANs with projection discriminator," in *International Conference on Learning Representations (ICLR)*, 2018.

[169] X. Huang, Y. Li, O. Poursaeed, J. Hopcroft, and S. Belongie, "Stacked generative adversarial networks," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017, pp. 5077–5086.

[170] T. Miyato, T. Kataoka, M. Koyama, and Y. Yoshida, "Spectral normalization for generative adversarial networks," in *International Conference on Learning Representations (ICLR)*, 2018.

[171] A. Brock, J. Donahue, and K. Simonyan, "Large scale GAN training for high fidelity natural image synthesis," in *International Conference on Learning Representations (ICLR)*, 2019.

[172] T. Karras, S. Laine, and T. Aila, "A style-based generator architecture for generative adversarial networks," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019, pp. 4401–4410.

[173] D. Bau, J. Zhu, J. Wulff, W. Peebles, H. Strobelt, B. Zhou, and A. Torralba, "Inverting layers of a large generator," in *International Conference on Learning Representations (ICLR)*, 2019.

[174] A. H. Li, Y. Wang, C. Chen, and J. Gao, "Decomposed adversarial learned inference," 2020. [Online]. Available: https://arxiv.org/abs/2004.10267

[175] C. Ledig, L. Theis, F. Huszár, J. Caballero, A. Cunningham, A. Acosta, A. Aitken, A. Tejani, J. Totz, Z. Wang, and W. Shi, "Photo-realistic single image super-resolution using a generative adversarial network," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017, pp. 105–114.

[176] A. Radford, L. Metz, and S. Chintala, "Unsupervised representation learning with deep convolutional generative adversarial networks," 2015. [Online]. Available: https://arxiv.org/abs/1511.06434

[177] T. Salimans, I. Goodfellow, W. Zaremba, V. Cheung, A. Radford, and X. Chen, "Improved techniques for training GANs," in *Conference and Workshop on Neural Information Processing Systems (NIPS)*, 2016, pp. 2234–2242.

[178] M. Heusel, H. Ramsauer, T. Unterthiner, B. Nessler, and S. Hochreiter, "GANs trained by a two time-scale update rule converge to a local Nash equilibrium," in *Conference and Workshop on Neural Information Processing Systems (NIPS)*, 2017, pp. 6629–6640.

[179] M. Lucic, K. Kurach, M. Michalski, O. Bousquet, and S. Gelly, "Are GANs created equal? A large-scale study," in *Conference and Workshop on Neural Information Processing Systems (NIPS)*, 2018, pp. 698–707.

[180] D. Scharstein and R. Szeliski, "A taxonomy and evaluation of dense two-frame stereo correspondence algorithms," *International Journal of Computer Vision*, vol. 47, no. 1-3, pp. 7–42, 2002.

[181] C. Strecha, R. Fransens, and L. Van Gool, "Combined depth and outlier estimation in multi-view stereo," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2006, pp. 2394–2401.

[182] X. Huang, "Cooperative optimization for energy minimization in computer vision: A case study of stereo matching," in *Pattern Recognition: 26th DAGM Symposium*, 2004, pp. 302–309.

[183] Z. Wang and Z. Zheng, "A region based stereo matching algorithm using cooperative optimization," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2008, pp. 1–8.

[184] Y. Boykov, O. Veksler, and R. Zabih, "Fast approximate energy minimization via graph cuts," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 23, no. 11, pp. 1222–1239, 2001.

[185] J. S. Yedidia, W. T. Freeman, and Y. Weiss, "Generalized belief propagation," in *Conference and Workshop on Neural Information Processing Systems (NIPS)*, 2000, pp. 689–695.

[186] M. J. Wainwright, T. S. Jaakkola, and A. S. Willsky, "Map estimation via agreement on trees: Message-passing and linear programming," *IEEE Transactions on Information Theory*, vol. 51, no. 11, pp. 3697–3717, 2005.

[187] R. Szeliski, R. Zabih, D. Scharstein, O. Veksler, V. Kolmogorov, A. Agarwala, M. Tappen, and C. Rother, "A comparative study of energy minimization methods for Markov Random Fields with smoothness-based priors," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 30, no. 6, pp. 1068–1080, 2008.

[188] F. Crow, "Summed-area tables for texture mapping," *Computer Graphics*, vol. 18, no. 3, pp. 207–212, 1984.

[189] L. Di Stefano, M. Marchionni, and S. Mattoccia, "A fast area-based stereo matching algorithm," *Image and Vision Computing*, vol. 22, no. 12, pp. 983–1005, 2004.

[190] E. Tola, V. Lepetit, and P. Fua, "DAISY: An efficient dense descriptor applied to wide-baseline stereo," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 32, no. 5, pp. 815–830, 2010.

[191] F. Tombari, S. Mattoccia, L. Di Stefano, and E. Addimanda, "Classification and evaluation of cost aggregation methods for stereo correspondence," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2008, pp. 1–8.

[192] H. Hirschmüller, "Stereo processing by semiglobal matching and mutual information," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 30, no. 2, pp. 328–341, 2008.

[193] K. J. Yoon and I. S. Kweon, "Adaptive support-weight approach for correspondence search," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 28, no. 5, pp. 650–656, 2006.

[194] S. Mattoccia, S. Giardino, and A. Gambini, "Accurate and efficient cost aggregation strategy for stereo correspondence based on approximated joint bilateral filtering," in *Asian conference on Computer Vision (ACCV)*, 2009, pp. 371–380.

[195] D. Min, J. Lu, and M. N. Do, "A revisit to cost aggregation in stereo matching: how far can we reduce its computational redundancy?" in *IEEE International Conference on Computer Vision (ICCV)*, 2011, pp. 1567–1574.

[196] A. Hosni, C. Rhemann, M. Bleyer, C. Rother, and M. Gelautz, "Fast cost-volume filtering for visual correspondence and beyond," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 35, no. 2, pp. 504–511, 2013.

[197] K. He, J. Sun, and X. Tang, "Guided image filtering," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 35, no. 6, pp. 1397–1409, 2013.

[198] Q. Yang, "A non-local cost aggregation method for stereo matching," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2012, pp. 1402–1409.

[199] X. Mei, X. Sun, W. Dong, H. Wang, and X. Zhang, "Segment-tree based cost aggregation for stereo matching," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2013, pp. 313–320.

[200] K. Zhang, Y. Fang, D. Min, L. Sun, S. Yang, S. Yan, and Q. Tian, "Cross-scale cost aggregation for stereo matching," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2014, pp. 1590–1597.

[201] M. Calonder, V. Lepetit, M. Ozuysal, T. Trzcinski, C. Strecha, and P. Fua, "BRIEF: Computing a local binary descriptor very fast," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 34, no. 7, pp. 1281–1298, 2012.

[202] S. Leutenegger, M. Chli, and R. Y. Siegwart, "BRISK: binary robust invariant scalable keypoints," in *IEEE International Conference on Computer Vision (ICCV)*, 2011, pp. 2548–2555.

[203] A. Alahi, R. Ortiz, and P. Vandergheynst, "FREAK: fast retina keypoint," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2012, pp. 510–517.

[204] E. Rublee, V. Rabaud, K. Konolige, and G. Bradski, "ORB: An efficient alternative to SIFT or SURF," in *IEEE International Conference on Computer Vision (ICCV)*, 2011, pp. 2564–2571.

[205] H. Bay, A. Ess, T. Tuytelaars, and L. Van Gool, "Speeded-up robust features (SURF)," *Computer Vision and Image Understanding*, vol. 110, no. 3, pp. 346–359, 2008.

[206] J. Heinly, E. Dunn, and J. M. Frahm, "Comparative evaluation of binary features," in *European Conference on Computer Vision (ECCV)*, 2012, pp. 759–773.

[207] N. Khan, B. McCane, and S. Mills, "Better than SIFT?" *Machine Vision and Applications*, vol. 26, no. 6, pp. 819–836, 2015.

[208] K. Zhang, J. Li, Y. Li, W. Hu, L. Sun, and S. Yang, "Binary stereo matching," in *International Conference on Pattern Recognition (ICPR)*, 2012, pp. 356–359.

[209] S. Zagoruyko and N. Komodakis, "Learning to compare image patches via convolutional neural networks," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015, pp. 4353–4361.

[210] J. Žbontar and Y. LeCun, "Computing the stereo matching cost with a convolutional neural network," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015, pp. 1592–1599.

[211] Z. Chen, X. Sun, L. Wang, Y. Yu, and C. Huang, "A deep visual correspondence embedding model for stereo matching costs," in *IEEE International Conference on Computer Vision (ICCV)*, 2015, pp. 972–980.

[212] W. Luo, A. G. Schwing, and R. Urtasun, "Efficient deep learning for stereo matching," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 5695–5703.

[213] Z. Liang, Y. Feng, Y. Guo, H. Liu, W. Chen, L. Qiao, L. Zhou, and J. Zhang, "Learning for disparity estimation through feature constancy," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018, pp. 2811–2820.

[214] G. Yang, H. Zhao, J. Shi, Z. Deng, and J. Jia, "SegStereo: Exploiting semantic information for disparity estimation," in *European Conference on Computer Vision (ECCV)*, 2018, pp. 660–676.

[215] M. L. Zhai, X. Z. Xiang, R. F. Zhang, N. Lv, and B. El Saddik, "Learning optical flow using deep dilated residual networks," *IEEE Access*, vol. 7, pp. 22 566–22 578, 2019.

[216] X. Song, X. Zhao, L. Fang, and H. Hu, "EdgeStereo: An effective multi-task learning network for stereo matching and edge detection," *International Journal of Computer Vision*, vol. 128, no. 2, pp. 910–930, 2020.

[217] N. Mayer, E. Ilg, P. Hausser, P. Fischer, D. Cremers, A. Dosovitskiy, and T. Brox, "A large dataset to train convolutional networks for disparity, optical flow, and scene flow estimation," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 4040–4048.

[218] C. Strecha, W. von Hansen, L. Van Gool, P. Fua, and U. Thoennessen, "On benchmarking camera calibration and multi-view stereo for high resolution

imagery," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2008, pp. 1–8.

[219] D. Scharstein, H. Hirschmller, Y. Kitajima, G. Krathwohl, N. Nesic, X. Wang, and P. Westling, "High-resolution stereo datasets with subpixel-accurate ground truth," in *German Conference on Pattern Recognition (GCPR)*, 2014, pp. 31–42.

[220] R. Zabih and J. Wood, "Non-parametric local transforms for computing visual correspondence," in *European Conference on Computer Vision (ECCV)*, 1994, pp. 151–158.

[221] H. Hirschmüller and D. Scharstein, "Evaluation of stereo matching costs on images with radiometric differences," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 31, no. 9, pp. 1582–1599, 2009.

[222] J. Banks and P. Corke, "Quantitative evaluation of matching methods and validity measures for stereo vision," *International Journal of Robotics Research*, vol. 20, no. 7, pp. 512–532, 2001.